Java开发利器





附：视频讲解
本书源代码

强锋科技
连洪武　编著

Eclipse Web开发从入门到精通(实例版)

- 作者有多年的Eclipse使用经验，开发过10多个大型项目
- 涉及Ant、Hibernate、JUnit、JBoss、Struts、CVS等多种开源项目
- 全书通过实例进行讲解，让读者在编码过程中不断进步
- 最后配多个大型案例，便于读者理解实际项目的开发
- 配多媒体视频演示光盘，讲解相关操作和典型配置



清华大学出版社

Java 开发利器

Eclipse Web 开发从入门到精通
（实例版）

强锋科技

连洪武　编著

清华大学出版社

北　京

内 容 简 介

本书由浅入深、循序渐进地介绍了目前流行的基于 Eclipse 的优秀框架。全书共分 14 章，内容涵盖 Eclipse 基础、Ant 资源构造、数据库应用开发、Web 应用开发、Struts 应用开发、Hibernate 应用开发、单元测试、AOP 和 CVS 等内容。最后还讲解了 3 个综合案例，具有较高的参考价值。本书最大的特色在于，每一节的例子都是经过精挑细选的，具有很强的针对性，力求让读者通过亲自动手尽可能快地掌握如何基于 Eclipse 进行企业应用开发，学习尽可能多的知识。

本书内容丰富，实例典型，适用于初、中级 Eclipse 用户，同时也可用作高校计算机专业和社会培训班的教材。

图书在版编目（CIP）数据

Eclipse Web 开发从入门到精通（实例版）/连洪武编著. —北京：清华大学出版社，2007.6
（Java 开发利器）
ISBN　978-7-302-15443-3

Ⅰ. E…　Ⅱ. 连…　Ⅲ. Java 语言—程序设计　Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2007）第 085445 号

责任编辑： 欧振旭　闫志朝
封面设计： 范华明
版式设计： 高　伟
责任校对： 马军令
责任印制：
出版发行： 清华大学出版社
http://www.tup.com.cn
c-service@tup.tsinghua.edu.cn
社 总 机： 010-62770175
投稿咨询： 010-62772015
地　　址： 北京清华大学学研大厦 A 座
邮　　编： 100084
邮购热线： 010-62786544
客户服务： 010-62776969
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 185×260　**印　　张：** 32.75　**字　　数：** 735　千字
（附光盘 1 张）
版　　次： 2007 年 6 月第 1 版　**印　　次：** 2007 年 6 月第 1 次印刷
印　　数： 1～6000
定　　价： 59.80 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：025067-01

前　　言

如今，Eclipse 越来越成为众多 Java 程序开发者首选的集成开发环境。层出不穷的插件和应用不断丰富着 Eclipse 的世界。在 SUN、IBM 等公司的积极推动下，“开源”之势在 Java 社区中日新月异，Hibernate、Struts、HSQLDB 等一大批优秀的开源框架脱颖而出，同时基于这些成熟框架而构建的成功企业应用也越来越多。

在实际的企业应用中，面对如此众多的优秀框架，通常需要解决两个问题：应该选择什么框架和如何使用这些框架。本书的目录是对第一个问题很好的回答，它总结了目前基于 Eclipse 平台的所有优秀框架，范围涉及从数据持久层、应用逻辑层到用户表示层的所有方面。我们每天都在不停地做着各种 DEMO，面对一个新的框架，第一步要做的是能调通一个 Hello World 级别的 DEMO。每当这时我们都希望能有一篇“step by step”的文章。

本书的目的就是力求让读者尽可能快地熟悉如何基于 Eclipse 平台进行企业应用开发。我们不求数学上的“一题多解”，但求寻找一种最快的方法解决问题。我们希望在本书的帮助下，让公司的新员工在三天之内熟悉 Eclipse 成为可能。

本书特点

1．实例讲解，易于学习

书中的每一章都精选了经典的实例进行讲解，每个实例都是一个完整的应用。

2．讲解通俗，步骤详细

本书的讲解始终贯穿着“跟我做”的思想，认真记录下键盘鼠标的每一个动作，加上丰富的插图和备注说明，让每个知识点都一目了然。

3．配视频演示光盘，加速学习

配书光盘中包含了所有的源代码，以方便读者使用。同时，光盘中还配带了作者专门制作的典型配置的多媒体演示，让读者快速入门。

4．完善的服务

作者的 E-mail 是 hongwulian@gmail.com，如果您在学习的过程中遇到什么困难，可以给作者发信，作者会及时回复。

本书内容

第 1 章带领读者了解 Eclipse 平台，包括下载并安装 Eclipse、Eclipse 的汉化，并且基于 Eclipse 完成了第一个 Java 实例。

第 2 章介绍 Eclipse 强大的代码重构功能。掌握这些技巧可以大大提高开发者的开发效率。

第 3 章介绍几个经典的 Eclipse 插件，包括 XML 文件的编写、直接查看字节码等基本内容。它对实际的应用开发很有益处。

第 4 章介绍如何在 Eclipse 中进行资源构建。内容包括 Eclipse 和 Ant 的集成，常用 Ant 操作等内容。

第 5 章以学生成绩管理系统为例，介绍如何基于 HSQLDB 进行数据库应用开发。

第 6 章以第 5 章为基础，介绍如何基于 JBoss 插件进行 Web 应用的开发。

第 7 章以在线留言板为例，介绍如何基于 Eclipse 开发 Struts 应用。

第 8 章以图书管理系统为例，介绍如何基于 Hibernate 框架进行应用开发。

第 9 章以第 8 章为基础，介绍如何使用 JUnit 框架对图书管理系统进行单元测试。

第 10 章详细介绍 JBossAOP 插件，并实现了第一个 AOP 实例。

第 11 章介绍 CVS 版本控制的使用。CVS 越来越成为团队开发不可或缺的工具。

第 12～14 章通过 3 个综合实例的开发全面应用本书涉及的开发技术，以达到进一步提高的目的。

读者对象

本书具有知识全面、实例典型、指导性强等特点，力求以全面的知识性及丰富的实例来指导读者透彻学习 Eclipse 各方面的技术。本书适合如下读者：

- 有一定 Java 基础的开发人员；
- JSP 开发人员；
- Web 开发人员；
- Eclipse 初、中级读者；
- J2EE 程序员；
- 培训学校的老师和学生；
- 大专院校的老师和学生。

本书作者

本书由强锋科技统筹，由连洪武编写。其他参与编写、资料整理及光盘制作的人员有王龙、王拥东、吴善才、徐砚颖、尹健慧、詹涵林、张薇、张小强、张运端、赵玉荣、郑慧、朱博、朱朝坤、邹小红、陈强、陈燕、丁凤霞、丁礼友、范忠诚、黄俊灿、贾伟、李喜彤、林垚、尚文谊、孙亮亮、唐崇敏、陶则熙等。在此对大家的辛勤工作一并表示感谢！

作者

2007 年 5 月

目　　录

第 1 篇　Eclipse 开发入门

第 2 篇　Web 开发技术实例详解

第 1 篇



Eclipse 开发入门

第 1 章　Eclipse 基础应用实例

Eclipse 是一个开放源代码的软件开发项目，专注于为高度集成的企业开发提供一个全功能的具有商业品质的工业平台。它最初由 IBM 公司贡献，IBM 公司提供了 Eclipse 代码基础。目前，由 IBM 牵头，Eclipse 项目已经发展成为一个庞大的联盟，有 150 多家软件公司参与到 Eclipse 项目中。其中包括 Borland、Rational Software、Red Hat 及 Sybase。最近 Oracle 也计划加入到 Eclipse 联盟中。

就其本身而言，Eclipse 只是一个框架和一组服务，用于通过插件组件构建开发环境。Eclipse 拥有一个标准的插件集，核心插件是 Platform、JDT 和 PDE。Platform 是 Eclipse 的核心运行平台，截至本书写作时的最新版为 3.1.1；JDT 是 Java 开发工具；PDE 是插件设计环境，用于设计自定义插件。正是这种基于插件的设计和开发方式，Eclipse 受到了越来越多的开发者的欢迎。

1.1　下载并安装 Eclipse

Eclipse 是开放源代码的项目，可以免费下载。它的官方网站的网址是 http：//www.eclipse.org 。官方网站提供 Releases、Stable Builds、Integration Builds 和 Nightly Builds 的下载。建议使用 Releases 或 Stable Builds 版本。Releases 版本是 Eclipses 开发团队发布的主要发行版本，是经过测试的稳定版本，适合要求稳定而不需要最新改进功能的使用者选择。目前最新的 Releases 版本是 Eclipse 3.2。Stable Builds 版本是对大多数使用者足够稳定的版本，由开发团队将认为比较稳定的 Integration Builds 版本提升到 Stable Builds 而来，适合想使用 Eclipse 新功能的使用者选择。

跟我做

1．安装 JDK 1.5

Eclipse 是一个基于 Java 平台的开发环境，它本身也要运行在 Java 虚拟机上，还要使用 JDK 的编译器，因此必须首先安装 JDK。

（1）登录 SUN 的官方站点 http://java.sun.com，下载 JDK 1.5 Windows 版。

（2）安装 JDK 1.5，可自行设定安装目录，如 E:\jdk15。

（3）设置系统变量 CLASSPATH。右击【我的电脑】，依次选择【属性】|【高级】|【环境变量】命令，弹出【环境变量】对话框，如图 1-1 所示。



图 1-1　【环境变量】对话框

（4）在【系统变量】中，单击【新建】按钮，弹出【新建系统变量】对话框，如图 1-2 所示。



图 1-2　新建系统变量

（5）新建系统变量。在【变量名】文本框中输入“JAVA_HOME”，在【变量值】文本框中输入 JDK 安装路径，如“E:\jdk15”，单击【确定】按钮。然后创建“CLASSPATH”系统变量，变量值为“.;E:\jre15\lib\rt.jar”。

（6）编辑“Path”系统变量。在【环境变量】对话框中选中“Path”系统变量，单击【编辑】按钮，弹出【编辑系统变量】对话框，在【变量值】文本框的最后加入“% JAVA_HOME %\bin”。

2．安装 Eclipse 3.1.1

（1）登录 Eclipse 的官方站点 http://www.eclipse.org，下载 Eclipse 安装包 eclipse-SDK-3.1.1-win32.zip。

（2）将 ZIP 文件解压至自行设定的安装目录，如 E:\eclipse，Eclipse 即安装完毕。

注意： 本书中 Eclipse 安装目录均用%Eclipse%代替。

1.2　安装语言包

Eclipse 提供了一个语言包的插件，用于国际化其开发环境。对于英文不好的读者或者

初学者来说，中文版本的 Eclipse 可以显著提高学习效率，便于接受和使用 Eclipse。在对 Eclipse 有了初步的认识之后，还是建议读者多使用英文版 Eclipse，这样对以后的进一步学习和使用有好处。

跟我做

（1）下载语言包插件。登录 Eclipse 语言包下载网站，网址是 http://download.eclipse.org/eclipse/downloads/drops/L-3.1.1_Language_Packs-200510051300/index.php。选择 SDK Language Packs 中的 NLpack1，下载 NLpack1-eclipse-SDK-3.1.1a-win32.zip。

（2）将 ZIP 文件解压至 Eclipse 安装路径，即可完成语言包的安装。

注意： 如果 Eclipse 环境不能完全汉化，可将%Eclipse%\configuration\org.eclipse.update 文件夹删除，重新启动 Eclipse 即可。如果想恢复英文环境，可增加-NL en_US 启动参数。

1.3　第一个 Java 实例

这是一个简单的 Java 程序，从命令行输入两个字符串参数，在程序中将其合并成一个字符串，并在控制台输出。本实例的目的是为了熟悉 Eclipse 环境，掌握编写 Java 程序的流程，掌握设置命令行参数的方法，以及如何在 Eclipse 中运行 Java 程序。

1.3.1　新建 Java 项目

（1）启动 Eclipse。右击%Eclipse%\ eclipse.exe，依次选择【发送到】|【桌面快捷方式】命令。

（2）双击桌面上的 Eclipse 快捷方式，启动 Eclipse。如 Eclipse 启动不成功，可增加虚拟机参数-vm %JAVA_HOME%\jre\bin\javaw.exe。

（3）单击【窗口】菜单，依次选择【打开透视图】|【Java】命令，打开 Java 透视图。

（4）单击【文件】菜单，依次选择【新建】|【项目】命令，弹出【新建项目】对话框。

（5）选择【Java 项目】，单击【下一步】按钮。

（6）在【项目名】文本框中输入“JavaApplication”，其他选项保持默认值，单击【完成】按钮。

1.3.2　配置构建路径

为了让 Java 源文件和编译后生成的字节码文件分开存放，需要配置构建路径。

（1）选择【窗口】|【首选项】命令，弹出【首选项】对话框。

（2）依次选择左侧列表的【Java】|【构建路径】，选中窗口中的文件夹选项，单击【确定】按钮，保存设置，如图 1-3 所示。

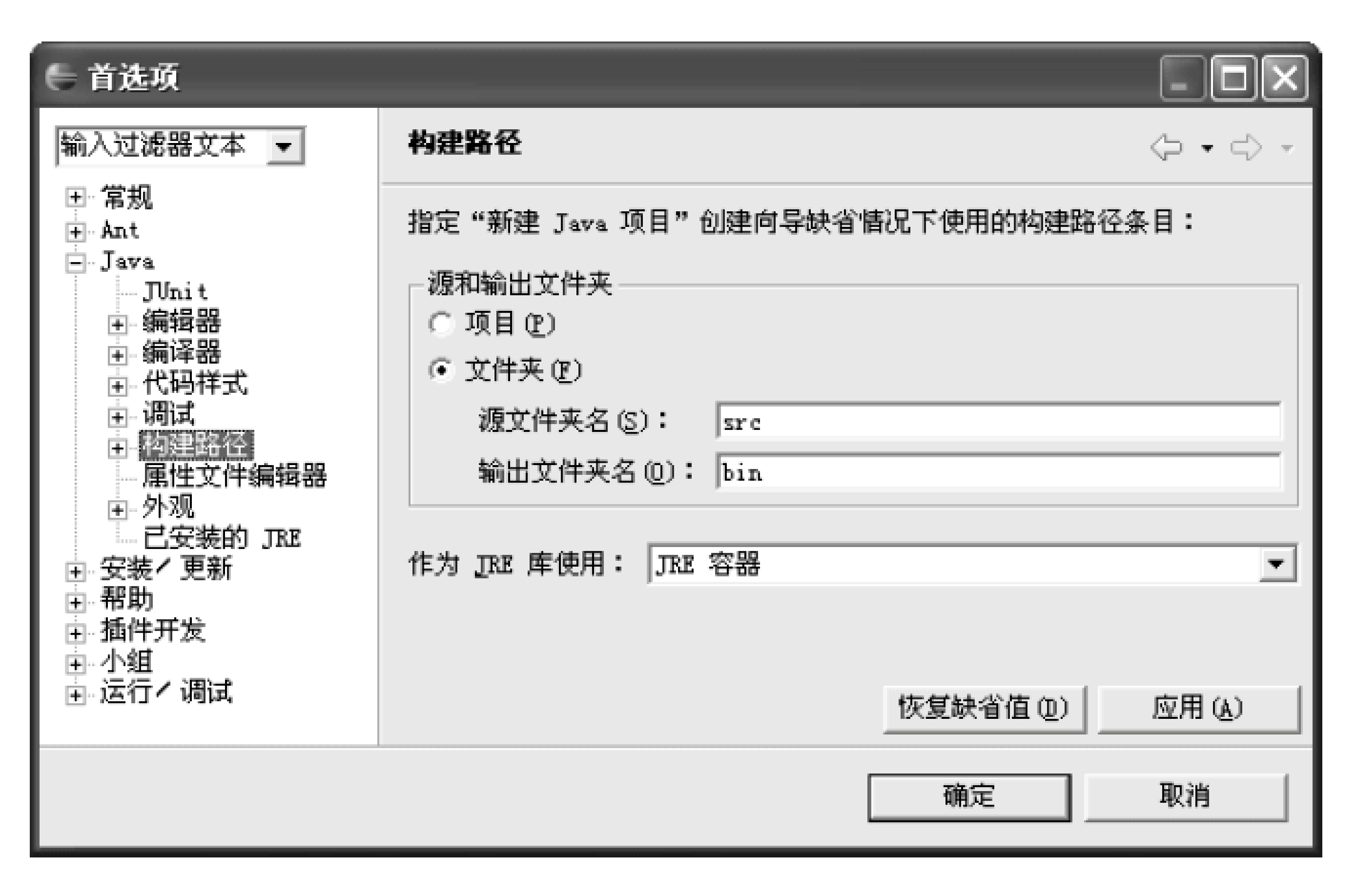


图 1-3　配置构建路径

1.3.3　新建 Java 类

（1）右击“src”文件夹，依次选择【新建】|【包】命令，弹出【新建 Java 包】对话框。在【名称】文本框中输入“net.chapter1”，单击【完成】按钮。

（2）右击“net.chapter1”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“StringJoin”，单击【完成】按钮。

（3）编写程序代码。如下：

```
package net.chapter1;
public class StringJoin {
    public static void main(String[] args) {
        //对输入的命令行参数进行判断，如果为空指针或者参数的数量不是 2 个，则输出提示信息后返回
        if (args == null || args.length != 2) {
            System.out.println("请输入两个字符串参数!");
            return;
        }
        //在控制台打印输入的参数
        System.out.println("您输入的字符串分别是:'" + args[0] + "'和'" + args[1] + "'");
        //将两个字符串进行连接
        String joinedString = args[0] + args[1];
        //在控制台打印连接后的字符串
        System.out.println("连接后的字符串是：" + joinedString);
    }
}
```

1.3.4　设置命令行参数

（1）右击“StringJoin.java”文件，依次选择【运行方式】|【运行...】命令，弹出【运行】对话框。

（2）在左侧【配置】列表中选择【Java 应用程序】，单击【新建】按钮，生成“StringJoin”项。

（3）选择右侧的【自变量】选项卡，在【程序自变量】文本框中输入命令行参数“hello”和“java”，如图 1-4 所示。

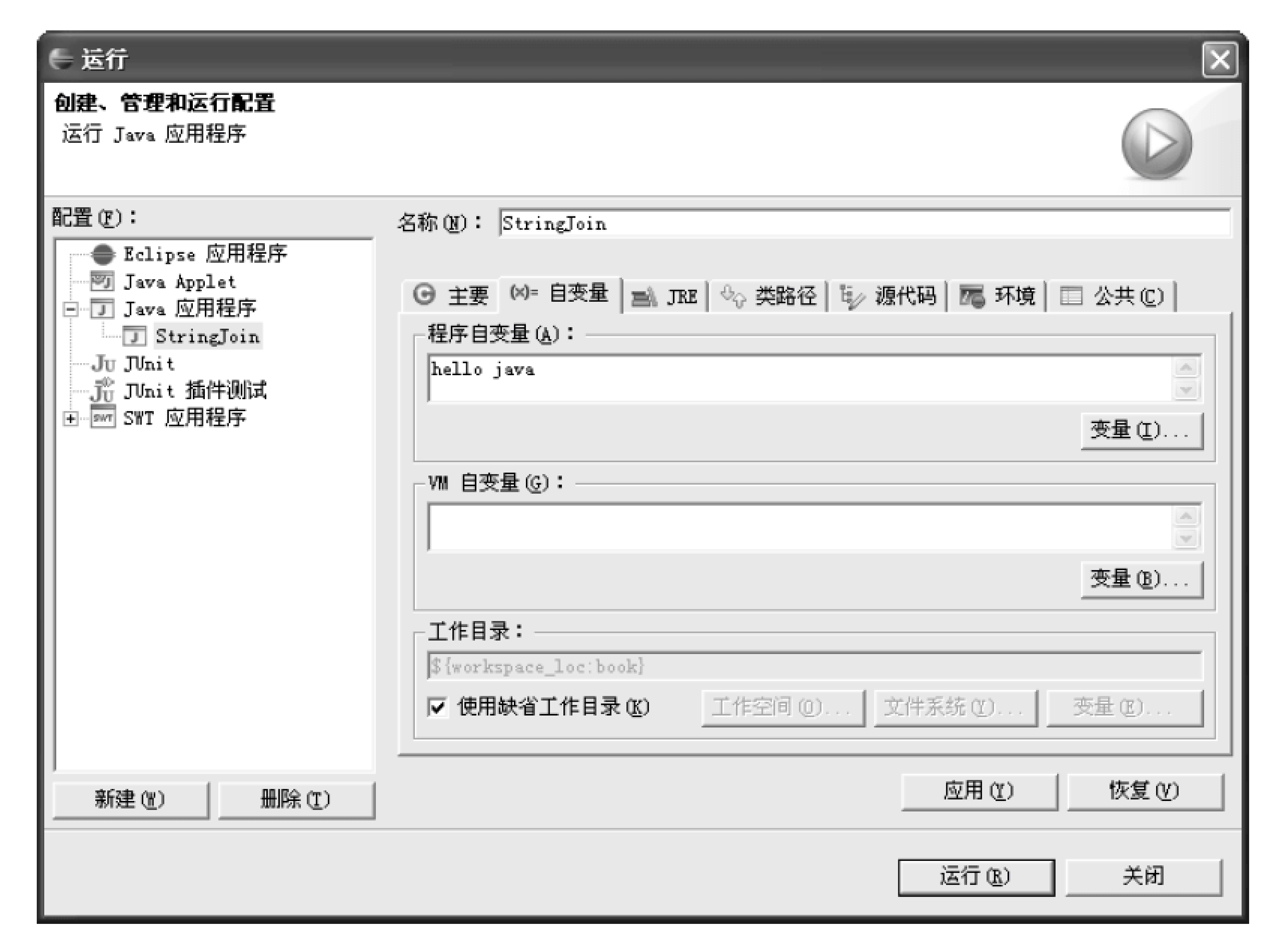


图 1-4　设置命令行参数

1.3.5　运行实例

在图 1-4 中，单击【运行】按钮运行 Java 程序。输出结果如图 1-5 所示。

```
您输入的字符串分别是:'hello'和'java'
连接后的字符串是：hellojava
```

图 1-5　第一个 Java 程序运行结果

1.4　Java 应用程序实例

本节将通过 3 个更详细的 Java 应用实例，进一步加深对 Eclipse 编程方式的理解。选择的题材包括排序算法、数字游戏、使用 Java 实现 FTP 功能。

1.4.1　排序算法的 Java 实现

排序问题应该是数据结构中的经典问题。前人总结出了多种排序算法，不同的算法都可以通过不同的语言描述实现。本实例将实现冒泡排序、插入排序、选择排序的 Java 版。实例运行后，控制台输出如图 1-6 所示。

```
原序为：45 67 199 -4 9 123 7 24 66 18

冒泡排序用时为：0
排序后：-4 7 9 18 24 45 66 67 123 199

插入排序用时为：0
排序后：-4 7 9 18 24 45 66 67 123 199

选择排序用时为：0
排序后：-4 7 9 18 24 45 66 67 123 199
```

图 1-6　3 种排序的输出

3 种排序结果一致，用时极短，均为 0。

跟我做

（1）右击 1.3 节创建的“net.chapter1”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“SortAlgorithm”，单击【完成】按钮。

（2）编写程序代码。

- 冒泡排序基本算法思想：从前到后扫描序列，比较相邻两个项目的大小，若发现逆序进行交换，最后使最大者换到序列的最后。然后再从后到前扫描剩下的序列，比较相邻两个项目的大小，若发现逆序则进行交换，最后使最小者换到序列的最前面。对剩下的序列重复上述过程，直到剩下的序列为空时止。bubbleSort 方法实现了冒泡排序。如下：

```
public class SortAlgorithm {
    private void bubbleSort(int[] numlist) {
        int out, in;
        // 从后到前，从倒数第二项到第二项扫描数列
        for (out = numlist.length - 1; out > 1; out--)
            // 从第一项到第 out 项扫描数列
            for (in = 0; in < out; in++)
                // 当前项大于后项，则交换两项的位置
                if (numlist[in] > numlist[in + 1]) {
                    int temp = numlist[in];
                    numlist[in] = numlist[in + 1];
                    numlist[in + 1] = temp;
                }
    }
```

- 插入排序基本算法思想：每次将一个待排序的数据元素插入到前面已经排好序的数列中的适当位置，使数列依然有序，直到待排序数据元素全部插入完为止。insertSort 方法实现了插入排序。如下：

```
    private void insertSort(int[] numlist) {
        int in, out;
        // 从 out 处分开数列
        for (out = 1; out < numlist.length; out++) {
            int temp = numlist[out];        // numlist[out]为待排序的数据
            in = out;
            while (in > 0 && numlist[in - 1] >= temp)       // 当发现数列值大于待排序的数据时
```

```
                {
                        numlist[in] = numlist[in - 1]; // 数列向右移一位
                        --in; // 指针向左移一位
                }
                numlist[in] = temp; // 插入待排序的数据
        }
}
```

- 选择排序基本算法思想：首先找出最小的元素，然后把这个元素与第一个元素互换，这样值最小的元素就放到了第一个位置。接着，再从剩下的元素中找值最小的，把它和第二个元素互换，使得第二小的元素放在第二个位置上，依此类推，直到所有的值由小到大排列为止。selectionSort 方法实现了选择排序。如下：

```
private void selectionSort(int[] numlist) {
        int out, in, min;
        // 从第一项到倒数第二项扫描数列
        for (out = 0; out < numlist.length - 1; out++) {
                min = out; // 将 out 项设为最小值
                // 从第二项到最后项扫描数列
                for (in = out + 1; in < numlist.length; in++)
                        // 当发现第 in 项小于最小值
                        if (numlist[in] < numlist[min])
                                min = in; // 将第 in 项作为最小值
                // 将 out 项与最小值进行交换
                int temp = numlist[out];
                numlist[out] = numlist[min];
                numlist[min] = temp;
        }
}
```

display 方法向控制台输出数列。如下：

```
private void display(int[] numlist) {
        for (int j = 0; j < numlist.length; j++)
                System.out.print(numlist[j] + " ");
        System.out.println("");
}
```

main 方法向控制台输出结果，首先输出原序，然后依次输出各种排序结果和排序时间。如下：

```
public static void main(String[] args) {
        SortAlgorithm sortAlgorithm = new SortAlgorithm();
        int numlist[] = new int[] { 45, 67, 199, -4, 9, 123, 7, 24, 66, 18 };        // 待排序数列
        // 打印待排序数列
        System.out.print("原序为：");
        sortAlgorithm.display(numlist);
        System.out.println("");

        long begin = System.currentTimeMillis();        // 排序开始时间，调用系统的当前时间
```

```
        sortAlgorithm.bubbleSort(numlist);                 // 执行冒泡排序
        long end = System.currentTimeMillis();             // 排序结束时间，调用系统的当前时间
        System.out.println("冒泡排序用时为：" + (end - begin));            // 排序用时
        // 打印排序结果
        System.out.print("排序后：");
        sortAlgorithm.display(numlist);

        System.out.println("");
        begin = System.currentTimeMillis();        // 排序开始时间
        sortAlgorithm.insertSort(numlist);         // 执行插入排序
        end = System.currentTimeMillis();          // 排序结束时间
        System.out.println("插入排序用时为：" + (end - begin));            // 排序用时
        // 打印排序结果
        System.out.print("排序后：");
        sortAlgorithm.display(numlist);
        System.out.println("");

        begin = System.currentTimeMillis();        // 排序开始时间
        sortAlgorithm.selectionSort(numlist);      // 执行选择排序
        end = System.currentTimeMillis();          // 排序结束时间
        System.out.println("选择排序用时为：" + (end - begin));            // 排序用时
        // 打印排序结果
        System.out.print("排序后：");
        sortAlgorithm.display(numlist);
    }
}
```

（3）运行实例。右击“SortAlgorithm.java”文件，依次选择【运行方式】|【Java 应用程序】命令。

1.4.2　猜数字游戏

游戏随机给出一个 0～100（包括 0 和 100）之间的数字，然后让游戏者猜是什么数字。游戏者可以随便猜一个数字，游戏会提示太大还是太小，从而缩小结果范围。经过几次猜测与提示后，最终推出答案。实例运行后，控制台输出如图 1-7 所示。

```
请输入0到100之间的整数：
50
您输入的数字大了，请再次输入：
25
您输入的数字小了，请再次输入：
38
您输入的数字小了，请再次输入：
45
您输入的数字小了，请再次输入：
48
您输入的数字大了，请再次输入：
47
您输入的数字大了，请再次输入：
46
答案正确。您共猜测7次。
```

图 1-7　猜数字游戏玩法

跟我做

（1）右击 1.3 节创建的“net.chapter1”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“GuessNumber”，单击【完成】按钮。

（2）编写程序代码。代码如下：

```
public class GuessNumber {
    public static void main(String[] args) {
        //新建一个随机数生成器
        Random random = new Random();
        //生成一个 0～100 之间的整数，方法 nextInt(int n)生成在 0（包括）和 n（不包括）之间均
匀分布的 int 值
        int randomNumber = random.nextInt(101);
        //记录游戏者所猜数字
        int guessedNumber = -1;
        //通过输入流取得游戏者从控制台输入的数字
        BufferedReader input = new BufferedReader(new InputStreamReader( System.in));
        int counter = 0;          //记录游戏者猜测的次数。
        System.out.println("");
        System.out.println("请输入 0 到 100 之间的整数：");
```

以下循环进行猜数字。如果输入的数字不在 0～100 的范围内，则提示输入值无效，再次输入。如果输入的不是数字，则抛出异常。代码如下：

```
        while (guessedNumber != randomNumber){
            try {
                //获取游戏者的输入
                guessedNumber = Integer.parseInt(input.readLine());
                if(guessedNumber<0 || guessedNumber>100){
                    System.out.println("请输入一个 0～100 之间的整数：");
                    counter++;          //游戏者猜测的次数加一
                    continue;
                }
            } catch (NumberFormatException e) {
                //如果游戏者输入的不是一个合法的整数，则让他重新输入
                System.out.println("请输入一个 0～100 之间的整数：");
                counter++;          //游戏者猜测的次数加一
                continue;
            } catch (IOException e) {
                System.out.println("程序发生异常错误将被关闭!");
                e.printStackTrace();
            }

            //对玩家的输入进行判断
            if (guessedNumber > randomNumber)
                System.out.println("您输入的数字大了，请再次输入：");
            if (guessedNumber < randomNumber)
                System.out.println("您输入的数字小了，请再次输入：");
```

```
                counter++;          //游戏者猜测的次数加一
            }
            System.out.println("答案正确。您共猜测"+counter+"次。");
        }
}
```

（3）运行实例。右击“GuessNumber.java”文件，依次选择【运行方式】|【Java 应用程序】命令。

1.4.3　通过 FTP 传递文件

本实例是一个使用 Java 实现 FTP 命令的程序，可从远程 FTP 服务器下载文件。程序中用到了 Apache 的 Commons Net 技术。它是一个用于操作 Internet 基础协议（Finger，Whois，TFTP，Telnet，POP3，FTP，NNTP 以及 SMTP）的底层 API。Commons Net 包不仅支持对各种低层次协议的访问，而且还提供了一个高层的抽象，使得开发者不再需要直接面对各种协议的 Socket 级的低层命令。本例只使用其 FTP 协议。

跟我做

（1）下载 Commons Net 包。可到 Apache 网站下载最新的 commons-net-1.4.1.jar，网址为 http://jakarta.apache.org/site/downloads/downloads_commons-net.cgi。

（2）右击“JavaApplication”项目，依次选择【新建】|【文件夹】命令，弹出【新建文件夹】对话框。在【文件夹名称】文本框中输入“lib”，单击【完成】按钮。将 commons-net-1.4.1.jar 复制到 lib 文件夹中。

（3）右击“JavaApplication”项目，选择【属性】命令，弹出【JavaApplication 的属性】对话框。选择【库】选项卡，单击【添加 Jar】按钮，弹出【选择 JAR】对话框，如图 1-8 所示。选中“commons-net-1.4.1.jar”，单击【确定】按钮。返回【JavaApplication 的属性】对话框，单击【确定】按钮。



图 1-8　【选择 JAR】对话框

（4）右击“net.chapter1”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“FTPUtil”，单击【完成】按钮。

（5）编写程序代码。如下：

```
public class FTPUtil {
    public static void main(String[] args) {
        //通过 FTPClient 的对象模拟用户访问 FTP 服务器
        FTPClient ftpClient = new FTPClient();
        String ftpServer = "101.10.10.24";                //FTP 服务器
        String remoteFile = "/opt/user/readme.txt";       //远程文件，复制的目标
        String localFile = "E:\\temp\\readme.txt";        //本地文件，复制的目的文件
        String user = "abc";                              //用户名
        String password = "edf";                          //密码
        try {
            ftpClient.connect(ftpServer);          //连接 FTP 服务器
            System.out.println("Connected to " + ftpServer + ".");
            int reply = ftpClient.getReplyCode();          //得到回复码
            //如果未能正常连接，退出程序
            if (!FTPReply.isPositiveCompletion(reply)) {
                System.out.println("FTP server refused connection!!");
                System.exit(1);
            }
            //通过用户名和密码登录 FTP 服务器，如果登录不成功，退出程序
            if (!ftpClient.login(user, password)) {
                ftpClient.logout();
                System.out.println("I can't login!!");
                System.exit(1);
            }

            //定义文件输出流，FTPClient 将把远程文件以流的方式输出，并写到本地文件
            OutputStream output = new FileOutputStream(localFile);
            ftpClient.retrieveFile(remoteFile, output); //如果传输成功，返回 true
            output.close(); //关闭流
            ftpClient.logout(); //推出登录
        } catch (IOException e) {
            System.err.println(e);
        } finally {
            //断开 FTP 连接
            if (ftpClient.isConnected()) {
                try {
                    ftpClient.disconnect();
                } catch (IOException f) {
                    System.err.println(e);
                }
            }
        }
    }
}
```

（6）运行程序。右击“FTPUtil.java”文件，依次选择【运行方式】|【Java 应用程序】命令。如果程序运行成功，将会把 FTP 服务器的/opt/user/目录下的 readme.txt 文件复制到本地 E:\temp 目录下。

1.5　SWT 界面开发实例

SWT（Standard Widget Toolkit，标准窗口小部件工具包）本身仅仅是 Eclipse 组织为了开发 Eclipse IDE 环境所编写的一组底层图形界面 API。或是无心插柳，或是有意为之，至今为止，SWT 在性能和外观上都超越了 SUN 公司提供的 AWT 和 SWING。

SWT 已经十分稳定，它最大化了操作系统的图形构件 API。也就是说，只要操作系统提供了相应图形的构件，SWT 就可以应用 JNI 技术调用它们，只有那些操作系统中不提供的构件，SWT 才自己去做一个模拟的实现。

1.5.1　使用 Shell 创建窗口

本小节将创建第一个 SWT 程序（注意，本章的例子主要针对 Windows 平台，其他操作系统大同小异）。本实例是一个简单的窗口，在窗口中间显示“你好，SWT!”字样。通过学习和运行本实例，读者将掌握配置 SWT API，编写和运行 SWT 程序的方法。

Display 负责监管 GUI 的资源并管理和操作系统的通信，它不仅要关注窗口是如何显示、移动和重画的，还要确保诸如鼠标点击、键盘敲击等事件送达小部件并去处理它们。Display 类不是可见的。

Shell 类扮演着 GUI 主窗口的角色。一个 Shell 实例是一个可视化的应用，对主窗口的打开、激活、最大化、最小化和关闭保持追踪。Shell 类的主函数为整合在 GUI 内的容器、小部件和事件提供了一个通用的接入点。从这一点讲，Shell 的作用像是这些组件的父类。本例的运行效果是一个简单的 SWT 窗口，如图 1-9 所示。



图 1-9　简单的 SWT 窗口

跟我做

本实例主要包括 3 部分，下面依次进行讲解。

1. 新建 SWT 项目

（1）单击【文件】菜单，依次选择【新建】|【项目】命令，弹出【新建项目】对话框。

（2）选择【Java 项目】，单击【下一步】按钮。

（3）在【项目名】文本框中输入“SWT”，其他选项保持默认值，单击【完成】按钮。

2. 引入 SWT 包

要编写 SWT 程序，需要引入 SWT 的 jar 包。Eclipse 组织并不提供单独的 SWT 包下载。必须下载完整的 Eclipse 开发环境才能得到 SWT 包。SWT 是作为 Eclipse 开发环境的一个插件形式存在。此插件名为$ECLIPSE\pluginsorg.eclipse.swt.win32.win32.x86_3.1.1.jar。

（1）右击“SWT”项目，选择【属性】命令，弹出【SWT 的属性】对话框。

（2）在左侧列表中选择【Java 构建路径】选项，在右侧单击【添加变量】按钮，弹出【新建变量路径条目】对话框。

（3）单击【配置变量】按钮，弹出【首选项】对话框。

（4）单击【新建】按钮，弹出【新建变量条目】对话框。

（5）在【名称】文本框中输入“SWT_LIB”，单击【文件】按钮，弹出【选择 Jar】对话框，选择$ECLIPSE\pluginsorg.eclipse.swt.win32.win32.x86_3.1.1.jar。单击【打开】按钮，返回【新建变量条目】对话框，依次单击【确定】按钮。

此时，SWT_LIB 变量已放入【构建路径上的 JAR 和类文件夹】列表框中，如图 1-10 所示。



图 1-10　SWT_LIB 的属性配置

3．编写程序

（1）右击“src”文件夹，依次选择【新建】|【包】命令，弹出【新建 Java 包】对话框。在【名称】文本框中输入“net.chapter4”，单击【完成】按钮。

（2）右击“net.chapter4”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“SimpleForm”，单击【完成】按钮。

（3）编写程序代码。代码如下：

```
public class SimpleForm {
    public static void main(String[] args) {
        Display display = new Display();          //创建 Display 实例
        Shell shell = new Shell(display);         //创建 Shell 实例
        shell.setText("窗口");                    //设置窗口的显示标签
        shell.setBounds(200,200,500,350);         //设置 shell 的显示范围
        //设置窗口布局
        FormLayout layout = new FormLayout();
        layout.marginHeight = 50;
        layout.marginWidth = 50;
```

```
        shell.setLayout(layout);
        //创建标签，用于显示"你好，SWT！"字样
        Label helloLabel=new Label(shell,SWT.CENTER); //采用 SWT.CENTER 样式，即居中
显示
        helloLabel.setText("你好，SWT！");
        shell.pack();                               //以紧凑方式显示窗口并自动调节大小
        shell.open();                               //打开 shell，类似于打开窗口
        //开始事件处理循环，直到用户关闭窗口
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

4．配置本机图形库并运行程序

用 SWT 编写的 GUI 与其所运行的平台的外观一致，速度也与本机程序相仿。这是因为 SWT 调用了操作系统自带的图形库，因此在运行程序时需指出本机图形库的位置。SWT 本机图形文件名一般为 swt-win32-nnnn.dll，这里的 nnnn 代表 4 位整数，它们位于 $ECLIPSE\plugins\org.eclipse.swt.win32. win32.x86_3.1.1.jar 包中。为了确保这些库文件能被程序所用，可将本机图形库放进 java.library.path 变量所包含的任何目录中。

（1）解压 org.eclipse.swt.win32.win32.x86_3.1.1.jar 至任意目录，如 E:\swt 目录。

（2）右击“SimpleForm.java”文件，依次选择【运行方式】|【运行】命令，弹出【运行】对话框。选择左侧的 SWT 应用程序，单击【新建】按钮，自动新建 SimpleForm 项。在右侧选择【自变量】选项卡，在【VM 自变量】文本框中输入“-Djava.library.path=E:\swt”，如图 1-11 所示。



图 1-11　java.library.path 参数配置

（3）单击【运行】按钮，运行 SimpleForm 程序。

1.5.2　简单的用户密码验证器

本小节实例将实现一个简单的用户密码验证器，如果输入用户名“abc”、密码“abc”，窗口将显示“通过”字样，否则显示“不通过”。

SWT 2.0 提供了一种新的标准布局：FormLayout。FormLayout 通过定义组件 4 个边的“粘贴”位置来排列组件，被引用的相对的组件可以是父组件，也可以是同一容器中的其他组件。

SWT 的事件处理模型和 AWT 类似，事件的处理委托给外部实体进行，实现了将事件源和监听器分开，通过 addXXXXListener 方法向组件注册监听器。本例将使用 SelectionAdapter 处理按钮单击事件。程序的运行效果分别如下。

（1）用户密码验证初始窗口如图 1-12 所示。

（2）当输入用户名“abc”、密码“abc”时，窗口显示“通过”字样，如图 1-13 所示。

（3）当输入其他的用户名和密码时，窗口显示“不通过”字样，如图 1-14 所示。



图 1-12　用户密码验证初始窗口



图 1-13　用户名密码通过验证



图 1-14　用户名密码未能通过验证

跟我做

1．新建 Java 文件 UserPassword.java

右击包名“net.chapter5”，依次选择【新建】|【类】命令，在弹出的新类对话框中输入类名 UserPassword，单击【完成】按钮。

2．编写代码

首先创建用于界面显示的控件，并设置 FormLayout 布局。代码如下：

```
public class UserPassword {
    Display display = new Display();       //创建 Display 实例
    Shell shell = new Shell(display);      //创建 Shell 实例
    Text nameText;                         //“用户名”文本框
    Text passText;                         //“密码”文本框
    Button submit;                         //“确定”按钮
    Button cancel;                         //“取消”按钮
    Label resultLabel;                     //显示验证结果标签
    //用户密码验证方法的实现
    public UserPassword() {
        //设置 FormLayout 布局,定义空白边
        FormLayout layout = new FormLayout();
```

```
FormData data = new FormData();   //定义具体的布局数据
layout.marginHeight = 20;          //设置空白边高度
layout.marginWidth = 30;           //设置空白边宽度
shell.setLayout(layout);
shell.setText("用户密码验证");

Label nameLabel = new Label(shell, SWT.NONE);          //“用户名”标签
nameLabel.setText("用户名");                           //设置标签内容
//定义 name 标签的位置。它的顶边离父组件（窗口 shell）的空白边距离是父组件 clientAr
（除空白边以外的空间）高度的 15%，偏移的点数为 0
data.top = new FormAttachment(15, 0);
nameLabel.setLayoutData(data);
```

定义 name 文本框的位置。它的顶边在 nameLabel 标签的中心位置，左边距 nameLabel 标签的右边有 10 个点。代码如下：

```
nameText = new Text(shell, SWT.SINGLE | SWT.BORDER);
data = new FormData();
data.top = new FormAttachment(nameLabel, 0, SWT.CENTER);
data.left = new FormAttachment(nameLabel, 10, SWT.RIGHT);
nameText.setLayoutData(data);
```

定义 passLabel 标签的位置。它的顶边距 nameLabel 标签的底边有 10 个点数的偏移。代码如下：

```
Label passLabel = new Label(shell, SWT.NONE);
passLabel.setText("密码");
data = new FormData();
data.top = new FormAttachment(nameLabel, 10, SWT.BOTTOM);
passLabel.setLayoutData(data);
```

定义 pass 文本输入的位置。它的顶边在 passLabel 标签的中心位置，左边与 nameText 文本框的左边对齐。代码如下：

```
passText = new Text(shell, SWT.SINGLE | SWT.BORDER);
passText.setEchoChar('*');//输入字符显示为"*"
passText.setTabs(2);
data = new FormData();
data.top = new FormAttachment(passLabel, 0, SWT.CENTER);
data.left = new FormAttachment(nameText, 0, SWT.LEFT);
passText.setLayoutData(data);

//两个 button 由一个使用 RowLayout 的组件来设置
Composite towButton = new Composite(shell, SWT.NONE);
RowLayout rowLayout = new RowLayout();
rowLayout.justify = true;
towButton.setLayout(rowLayout);
```

定义 button 组件的位置。它的顶边距 passLabel 标签的底边有 15 个点数，左边与 passLabel 标签的左边对齐，右边与 passText 文本输入的右边对齐。代码如下：

```
data = new FormData();
data.top = new FormAttachment(passLabel, 15, SWT.BOTTOM);
data.left = new FormAttachment(passLabel, 0, SWT.LEFT);
data.right = new FormAttachment(passText, 0, SWT.RIGHT);
towButton.setLayoutData(data);
```

定义显示结果的 resultLabel 标签。它的顶边距 towButton 的底边有 15 个点数，左边与 towButton 标签的左边对齐，右边与 towButton 文本输入的右边对齐。代码如下：

```
Composite resultComp = new Composite(shell, SWT.NONE);
resultComp.setLayout(rowLayout);
resultLabel = new Label(resultComp, SWT.CENTER);
resultLabel.setText("是否通过验证？");
data = new FormData();
data.top = new FormAttachment(towButton, 15, SWT.BOTTOM);
data.left = new FormAttachment(towButton, 0, SWT.LEFT);
data.right = new FormAttachment(towButton, 0, SWT.RIGHT);
resultComp.setLayoutData(data);
```

为 submit 按钮添加事件监听器，如果用户名和密码均为“abc”，则通过验证，否则不通过。代码如下：

```
submit = new Button(towButton, SWT.PUSH);
submit.setText("确定");
submit.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        if (nameText.getText().equals("abc")//判断用户名是否为“abc”
                && passText.getText().equals("abc")) {//判断密码是否为“abc”
            resultLabel.setText("通过");
        } else {
            resultLabel.setText("不通过");
        }
    }
});
```

为 cancel 按钮添加事件监听器，将 nameText 文本框和 passText 文本框清空。代码如下：

```
cancel = new Button(towButton, SWT.PUSH);
cancel.setText("取消");
cancel.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent event) {
        nameText.setText("");
        passText.setText("");
    }
});
```

```
        shell.pack();
        shell.open();
        //开始事件处理循环，直到用户关闭窗口
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch()) {
                // If no more entries in event queue
                display.sleep();
            }
        }
        display.dispose();
    }
    //主函数
    public static void main(String[] args) {
        new UserPassword();
    }
}
```

3．设置 java.library.path 参数并运行程序

（1）右击“UserPassword.java”文件，依次选择【运行方式】|【运行】命令，弹出【运行】对话框。选择左侧的 SWT 应用程序，单击【新建】按钮，自动新建 UserPassword 项。在右侧选择【自变量】选项卡，在【VM 自变量】文本框中输入“-Djava.library.path=E:\swt”。

（2）单击【运行】按钮，运行 UserPassword 程序。

1.5.3　文件选择器

本小节实例将实现一个文件选择器。从文件选择器中可选择一个或多个文件，选择完成后，可在窗口中显示所选文件的路径和名称。FileDialog 类实现了用户对文件的操作，它可让用户浏览文件，选择一个或多个文件。用户可设置默认的文件浏览路径，可设置默认的文件类型。程序的运行效果分别如下。

（1）文件选择器运行初始窗口如图 1-15 所示。

（2）单击【请选择文件】按钮，弹出【打开】对话框，如图 1-16 所示。



图 1-15　文件选择器运行初始窗口

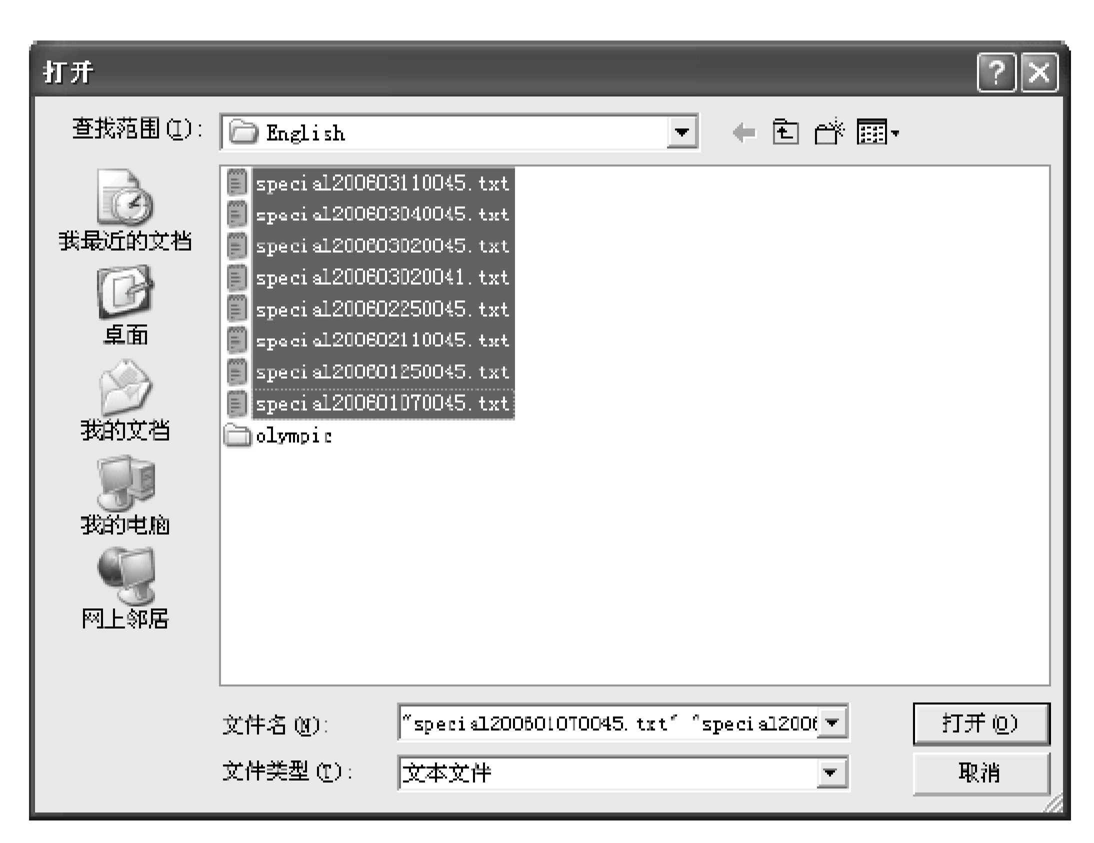


图 1-16　【打开】对话框

（3）选择多个文件后，将在【文件选择器】窗口显示所选文件的路径和名称，如图 1-17 所示。



图 1-17　选择文件后的窗口

跟我做

1．新建 Java 文件 FileSelection.java

右击包名“net.chapter5”，依次选择【新建】|【类】命令，在弹出的新建类对话框中输入类名 FileSelection，单击【完成】按钮。

2．编写代码

```
public class FileSelection {
    Display display = new Display();        //创建 Display 实例
    Shell shell = new Shell(display);       //创建 Shell 实例
    Label fileLabel;                        //用于显示文件的标签
    Button selectFileButton;                //文件选择按钮
    String filePath = "c:\\";               //“打开”文件对话框默认路径
    //文件选择器的实现
    public FileSelection() {
        //设置标签的样式、背景和文字
        fileLabel = new Label(shell, SWT.BORDER | SWT.WRAP);
        fileLabel.setBackground(display.getSystemColor(SWT.COLOR_WHITE));
        fileLabel.setText("单击按钮，选择文件。");
        //实例化按钮并设置显示文字
        selectFileButton = new Button(shell, SWT.PUSH);
        selectFileButton.setText("请选择文件：");
```

以下为程序的关键部分：为按钮添加事件监听器，限定默认文件的类型为文本文件和 HTML 文件。用户选择多个文件后，将文件名放入一个 String 类型的数组中。代码如下：

```
        selectFileButton.addListener(SWT.Selection, new Listener() {
            public void handleEvent(Event event) {
                //设置弹出的文件选择对话框
                FileDialog fileDialog = new FileDialog(shell, SWT.MULTI);
                fileDialog.setFilterPath(filePath);
                //限定文件的类型
                fileDialog.setFilterExtensions(new String[] { "*.txt","*.html", "*.*" });
                fileDialog.setFilterNames(new String[] { "文本文件", "HTML 文件", "任何" });
                String firstFile = fileDialog.open();
                //当选择了一或多个文件
                if (firstFile != null) {
                    filePath = fileDialog.getFilterPath();
```

```
                //取得文件名并在标签中显示
                String[] selectedFiles = fileDialog.getFileNames();
                StringBuffer sb = new StringBuffer("您选择了如下的文件 "
                        + fileDialog.getFilterPath() + "\\ \n");
                for (int i = 0; i < selectedFiles.length; i++) {
                    sb.append(selectedFiles[i] + "\n");
                }
                fileLabel.setText(sb.toString());
            }
        }
    });
    fileLabel.setBounds(0, 0, 400, 100);
    selectFileButton.setBounds(100, 110, 200, 25);
    shell.setText("文件选择器");
    shell.pack();
    shell.open();
    //开始事件处理循环，直到用户关闭窗口
    while (!shell.isDisposed()) {
        if (!display.readAndDispatch()) {
            // If no more entries in event queue
            display.sleep();
        }
    }
    display.dispose();
  }
  //主函数
  public static void main(String[] args) {
      new FileSelection();
  }
}
```

3．设置 java.library.path 参数并运行程序

（1）右击“FileSelection.java”文件，依次选择【运行方式】|【运行】命令，弹出【运行】对话框。选择左侧的 SWT 应用程序，单击【新建】按钮，自动新建 FileSelection 项。在右侧选择【自变量】选项卡，在【VM 自变量】文本框中输入“-Djava.library.path=E:\swt”。

（2）单击【运行】按钮，运行 FileSelection 程序。

第 2 章　在 Eclipse 中进行重构

重构，即改善现有代码的设计，是指在不改变代码的外部行为情况下而修改源代码。重构支持的目标是改进代码而不更改其行为。Eclipse 的 JDT 具有自动管理重构的功能。Eclipse 的重构可以分为 3 大类：

- 对代码进行重命名以及改变代码的物理结构，包括对属性、变量、类以及接口重新命名，还有移动包和类等。
- 改变类一级的代码逻辑结构，包括将匿名类转变为嵌套类，将嵌套类转变为顶级类，根据具体的类创建接口，从一个类中将方法或者属性移到子类或者父类中。
- 改变一个类内部的代码，包括将局部变量变成类的属性，将某个方法中选中部分的代码变成一个独立的方法，以及为属性生成 getter 和 setter 方法。

Eclipse 允许在最终选择执行重构前预览重构操作的所有即将产生的结果，在重构后也可撤销已经完成的重构。

2.1　重命名实例

“重命名”重构是指对所选择的元素进行命名更正，并更正对该元素的所有引用。该重构可用于方法、方法参数、字段、局部变量、类型、类型参数、枚举常量、编译单元、包、源文件夹和项目等元素的重命名。

跟我做

1．准备重构代码

为了演示 Eclipse 的重构功能，新建一个 Worker 类，本章的重构实例均基于此类。

（1）右击“src”文件夹，依次选择【新建】|【包】命令，弹出【新建 Java 包】对话框。在【名称】文本框中输入“net.chapter2”，单击【完成】按钮。

（2）右击“net.chapter2”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“Worker”，单击【完成】按钮。

（3）编写程序代码。代码如下：

```
package net.chapter2;
public class Worker {
	int id;			//员工编号
	String name;	//员工姓名
	//构造函数
	Worker() {
```

```
	}
	//构造函数
	Worker(int id, String name) {
		this.id = id;
		this.name = name;
	}
	//员工编号的 getter/setter
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
	//员工姓名的 getter/setter
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
}
```

再编写一个 Util.java，对 Worker.java 进行操作。代码如下：

```
public class Util{
	public static void main(String[] args) {
		Worker worker = new Worker(1,"John");
		System.out.println(worker);
	}
}
```

2．重构类文件名

（1）右击“Worker.java”文件，依次选择【重构】|【重命名】命令，弹出【重命名编译单元】对话框。

（2）在【新名称】文本框中输入“Worker1”，如图 2-1 所示。



图 2-1　重命名 Worker.java

❑ 【更新引用】复选框是指更新项目中所有引用“Worker.java”文件的 Java 文件，

如 import 语句等。

❑ 【更新注释和字符串中的文本匹配项】复选框是指更新项目中所有含有“Worker”字样的注释和字符串。

❑ 【更新非 Java 文件中的标准名称】复选框是指更新项目中所有非 Java 文件的“Worker”字样。

（3）要预览重命名导致的更改，单击【预览】按钮，如图 2-2 所示。Util 类中对 Worker 的引用已经变成了 Worker1。

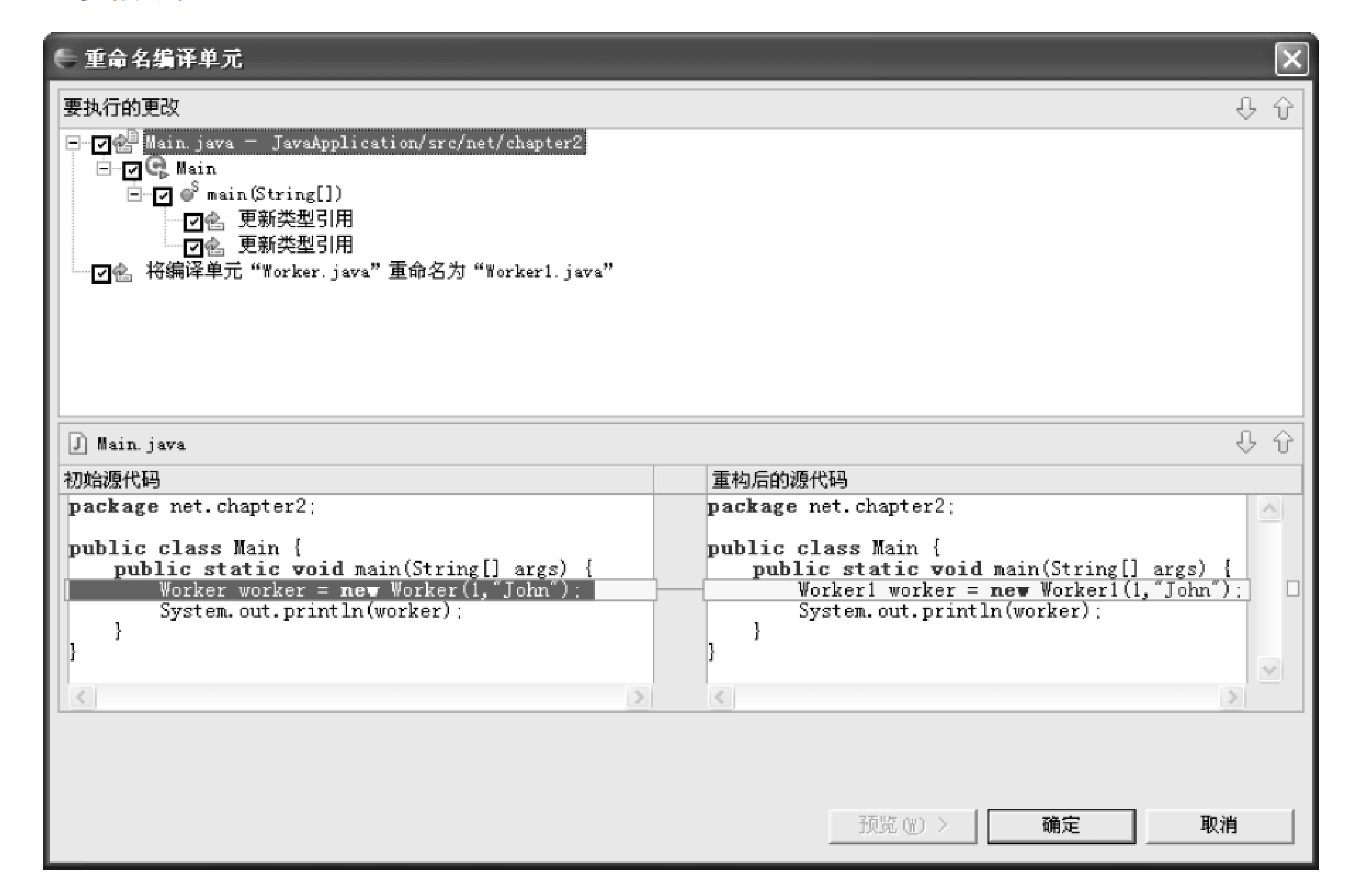


图 2-2　预览重命名导致的更改

（4）单击【确定】按钮，完成重命名。

（5）若要撤销重构，单击【编辑】菜单，选择【撤销 重命名】命令即可；也可直接按快捷键 Ctrl+Z。

2.2　移动实例

“移动”重构是指移动所选择的元素，并更正对元素的所有引用。该重构可用于一个实例方法、一个或多个静态方法、静态字段、类型、编译单元、包、源文件夹和项目的移动。

跟我做

（1）将“Worker.java”移动到“net.chapter1”包中。右击“Worker.java”文件，依次选择【重构】|【移动】命令，弹出【移动】对话框，如图 2-3 所示。

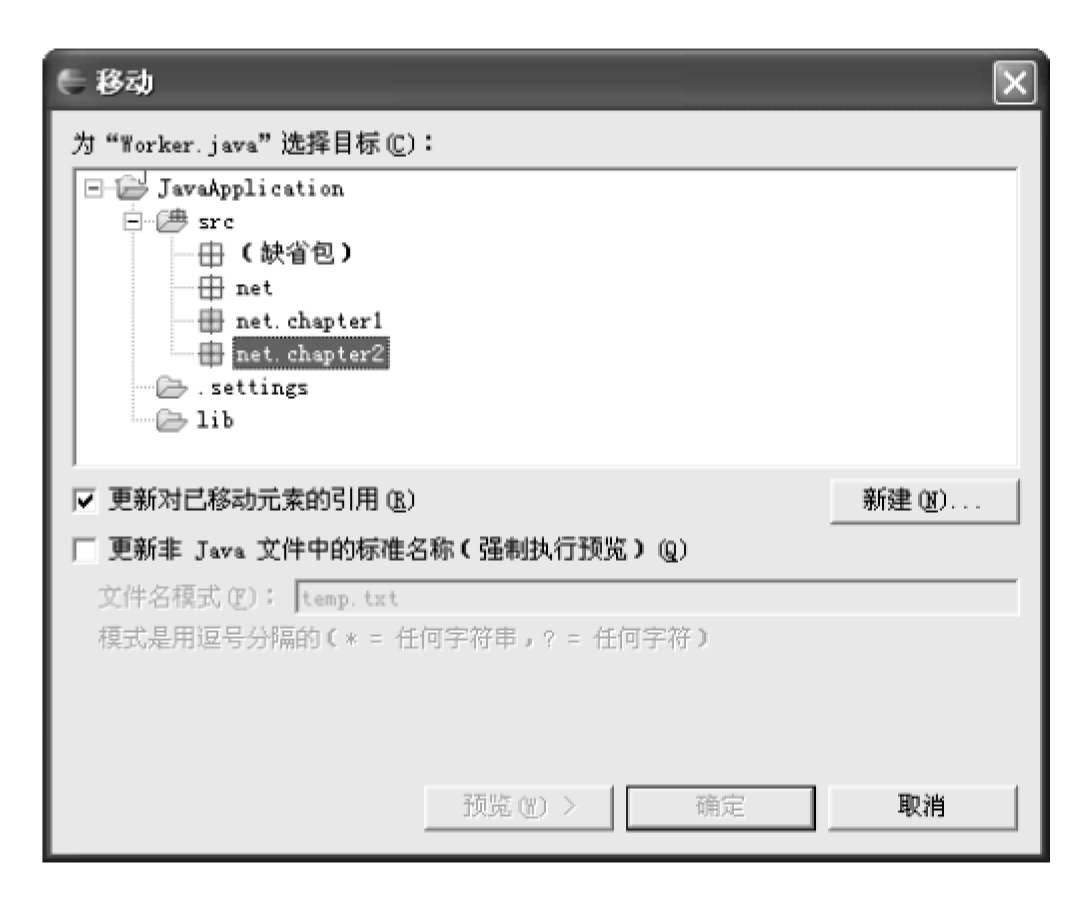


图 2-3　移动“Worker.java”

（2）选中“net.chapter2”包，单击【预览】按钮，预览移动导致的更改，如图 2-4 所示。

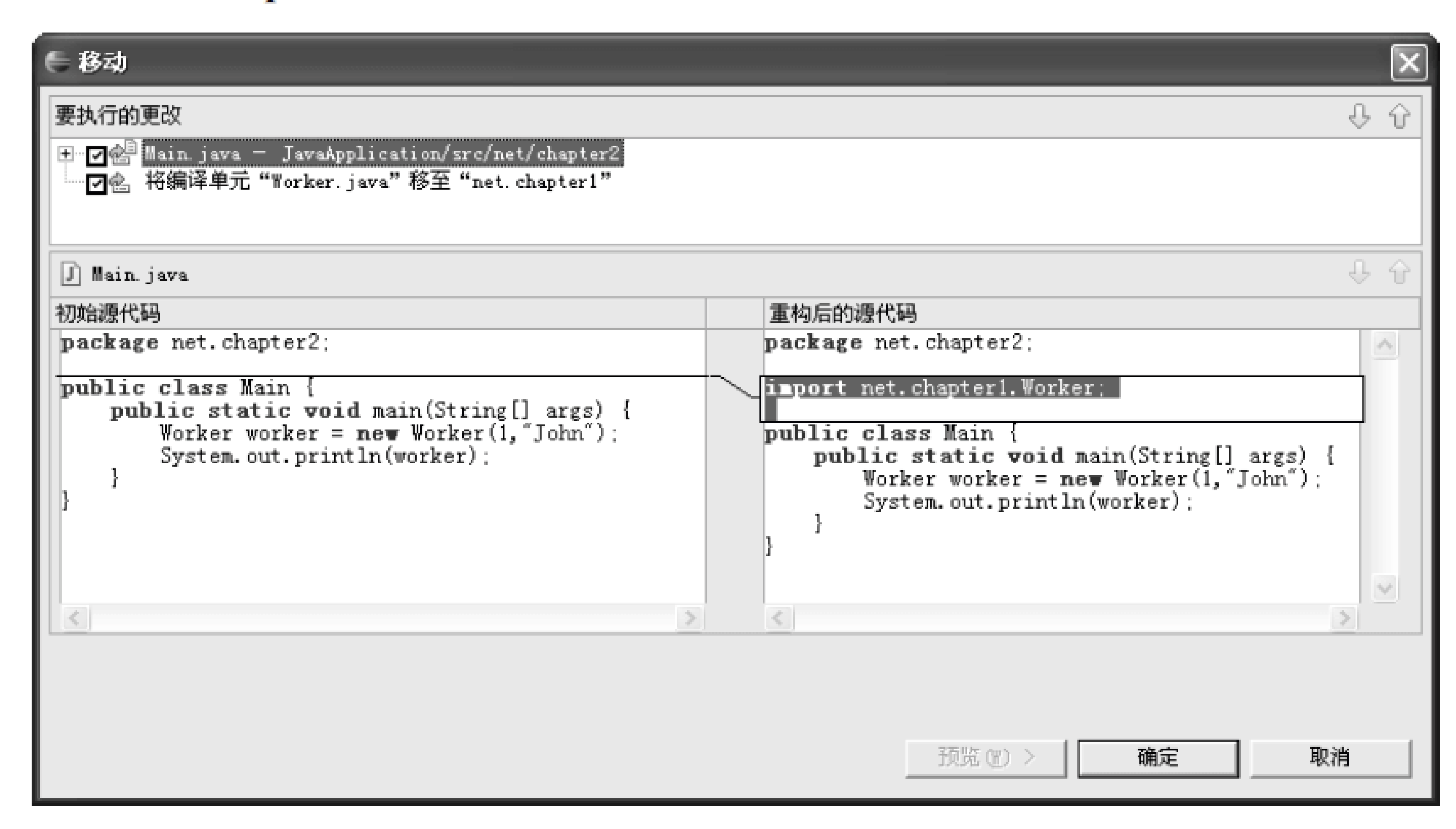


图 2-4　预览移动导致的更改

Util 类中增加了“import net.chapter1.Worker;”语句。

（3）单击【确定】按钮，完成移动。

（4）按快捷键 Ctrl+Z，撤销重构。

2.3　更改方法特征符实例

“更改方法特征符”重构可以更改方法的参数名称、参数类型和参数顺序，并更新对相应方法的所有引用。此外，可以除去或添加参数，并且可更改方法返回类型和它的可视性。该重构可用于方法或解析为方法的文本选择。

跟我做

（1）为了说明更改方法特征符重构功能，在 Util.java 中增加一个方法，用来计算员工的薪水。代码如下：

```
/**
 * 计算员工薪水
 * @param id   员工编号
 * @return   员工薪水
 */
public int countSalary(int id) {
    int wage = 3000;                //工资
    int bonus = 1500;               //奖金
    int subsidy = 1000;             //津贴
    return wage + bonus + subsidy;        //员工薪水是工资、奖金和津贴的总和
}
```

（2）双击“countSalary”方法名，选中方法。单击【重构】菜单，选择【更改方法特征符】选项，弹出【更改方法特征符】对话框，如图 2-5 所示。



图 2-5　更改“countSalary”方法特征符

（3）单击【访问修饰符】下拉列表框，选择【private】选项，在【方法名称】文本框中输入“getSalary”，在【重构】选项卡中单击【除去】按钮，删除原来的参数“id”。单击【添加】按钮，在【类型】项输入“String”，在【名称】项输入“name”。可以在【方法特征符预览】标签中看到方法特征符已经改变。

（4）单击【预览】按钮，查看方法特征符的改变，如图 2-6 所示。

（5）单击【确定】按钮，完成方法特征符的改变。改变后的代码如下：

```
/**
 * 计算员工薪水
 * @param name   员工姓名
 * @return   员工薪水
 */
public int countSalary(String name) {
    int wage = 3000;                //工资
    int bonus = 1500;               //奖金
    int subsidy = 1000;             //津贴
    return wage + bonus + subsidy;        //员工薪水是工资、奖金和津贴的总和
}
```

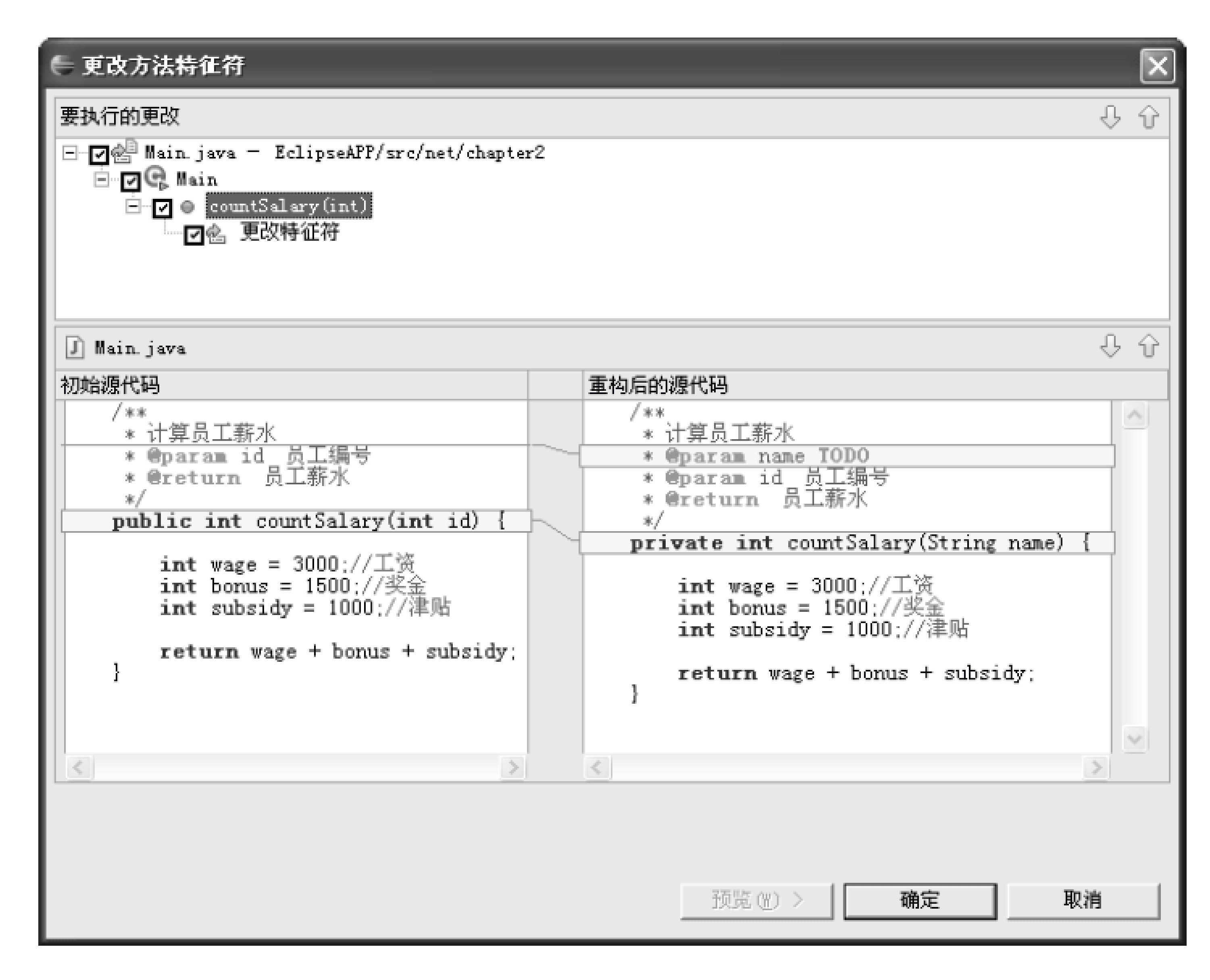


图 2-6　预览方法特征符的改变

2.4　将匿名类转换为嵌套类实例

“将匿名类转换为嵌套类”重构可将匿名类转换为嵌套类（内部类）。该重构只在 Java 文件内部使用，2.5 节将演示将内部类移至新的 Java 文件。

跟我做

（1）为了说明将匿名类转换为嵌套类重构功能，在 Util.java 中增加匿名类，它只有一个方法，用于将输入的参数输出到控制台。代码如下：

```
    Object obj = new Object() {              //匿名类声明为 Object 的对象
        public void printName(String name) {
            System.out.println(name);
        }
};              //注意最后的分号是必需的
```

（2）将光标定位到匿名类的内部，单击【重构】菜单，选择【将匿名类转换为嵌套类】命令，弹出【将匿名类转换为嵌套类】对话框，如图 2-7 所示。



图 2-7　【将匿名类转换为嵌套类】对话框

（3）在【访问修饰符】中选中【public】单选按钮，在【类名】文本框中输入“InnerClass”，并可将生成的嵌套类声明为“终态”和“静态”。

（4）单击【预览】按钮，查看 Util 类改变，如图 2-8 所示。



图 2-8　预览匿名类转换为嵌套类

（5）单击【确定】按钮，完成匿名类转换为嵌套类的重构。改变后的代码如下：

```
//新生成的嵌套类
public class InnerClass {
    //printName 方法没有改变
    public void printName(String name) {
        System.out.println(name);
    }
}
Object obj = new InnerClass();        //Object 对象引用 InnerClass 的实例
```

2.5　将成员类型移至新文件实例

“将成员类型移至新文件”重构可将所选成员类型创建新的 Java 编译单元，并根据需要更新所有引用。对于非静态成员类型来说，将添加字段以允许在必要时访问先前的外层实例。该重构可用于成员类型或解析为成员类型的文本。

跟我做

（1）2.4 节将匿名类转换为了嵌套类，本节将生成的嵌套类创建为新的 Java 类文件。嵌套类代码如下：

```
//嵌套类
public class InnerClass {
    public void printName(String name) {
        System.out.println(name);
    }
}
```

（2）将光标定位到嵌套类的内部，单击【重构】菜单，选择【将成员类型移至新文件】命令，弹出【将成员类型移至新文件】对话框，如图 2-9 所示。



图 2-9　【将成员类型移至新文件】对话框

（3）直接单击【预览】按钮，查看改变。Util.java 的变化如图 2-10 所示。



图 2-10　Util.java 的改变

（4）在【要执行的更改】中选中【创建文件】复选框，预览新增加的 InnerClass.java 文件，如图 2-11 所示。



图 2-11　新增的 InnerClass.java 文件

（5）单击【确定】按钮，完成重构。在【包资源管理器】中可以看到“net.chapter2”包增加了 InnerClass.java 文件。可用“重命名”重构方法，将其重命名为“OuterClass.java”。

2.6　上拉实例

“上拉”重构可将字段或方法移至其声明类的超类或者（对于方法）将方法声明为超类中的抽象类。该重构可用于在同一个类型中声明的一个或多个方法、字段和成员类型，也可以应用于字段、方法或成员类型内的文本选择。

跟我做

（1）为了说明“上拉”重构功能，新建一个 Employee.java 类，作为 Worker.java 的超类。代码如下：

```
public class Employee {
      Employee(){
      }
}
```

（2）修改 Worker.java 代码，使其继承自 Employee.java。代码如下：

```
public class Worker extends Employee{
      …           //类的主体
}
```

（3）将 Worker.java 中的 id 属性及其 getter/setter 方法上拉至 Employee.java。将光标定位到 Worker.java 的内部，单击【重构】菜单，选择【上拉】命令，弹出【重构】对话框，如图 2-12 所示。



图 2-12　上拉 id 属性及其 getter/setter

（4）在【选择目标类】中选择“net.chapter2.Employee”，在【为成员指定操作】列表中选中【id】、【getId()】、【setId(int) 】复选框，单击【完成】按钮。

（5）查看 Worker.java 的源代码，id 属性及其 getter/setter 方法已经消失了。在 Employee.java 中增加了 id 属性及其 getter/setter 方法。代码如下：

```
//上拉后的 Employee 类
public class Employee {
	Employee() {
	}
	protected int id;
	public int getId() {
		return id;
	}
	public void setId(int id) {
		this.id = id;
	}
}
```

2.7　下推实例

“下推”重构可将一组方法和字段从一个类移至它的子类。该重构可用于在同一个类型中声明的一个或多个方法和字段或者字段或方法内的文本选择。

跟我做

（1）本实例将 2.6 节中上拉到 Employee.java 中的 id 属性及其 getter/setter 方法下推回 Worker.java。将光标定位到 Employee.java 的内部。

（2）单击【重构】菜单，选择【下推】命令，弹出【下推】对话框，如图 2-13 所示。

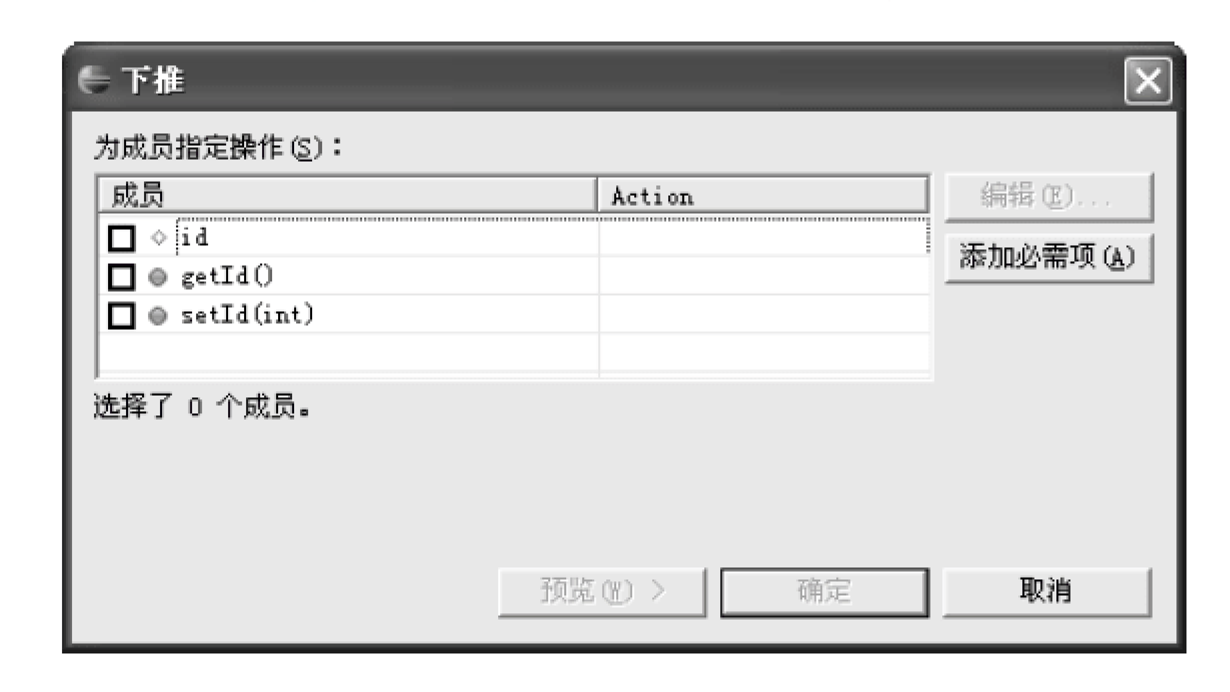


图 2-13　下推 id 属性及其 getter/setter 方法

（3）在【为成员指定操作】列表中选中【id】、【getId()】、【setId(int)】复选框，单击【预览】按钮。Employee.java 的变化如图 2-14 所示。

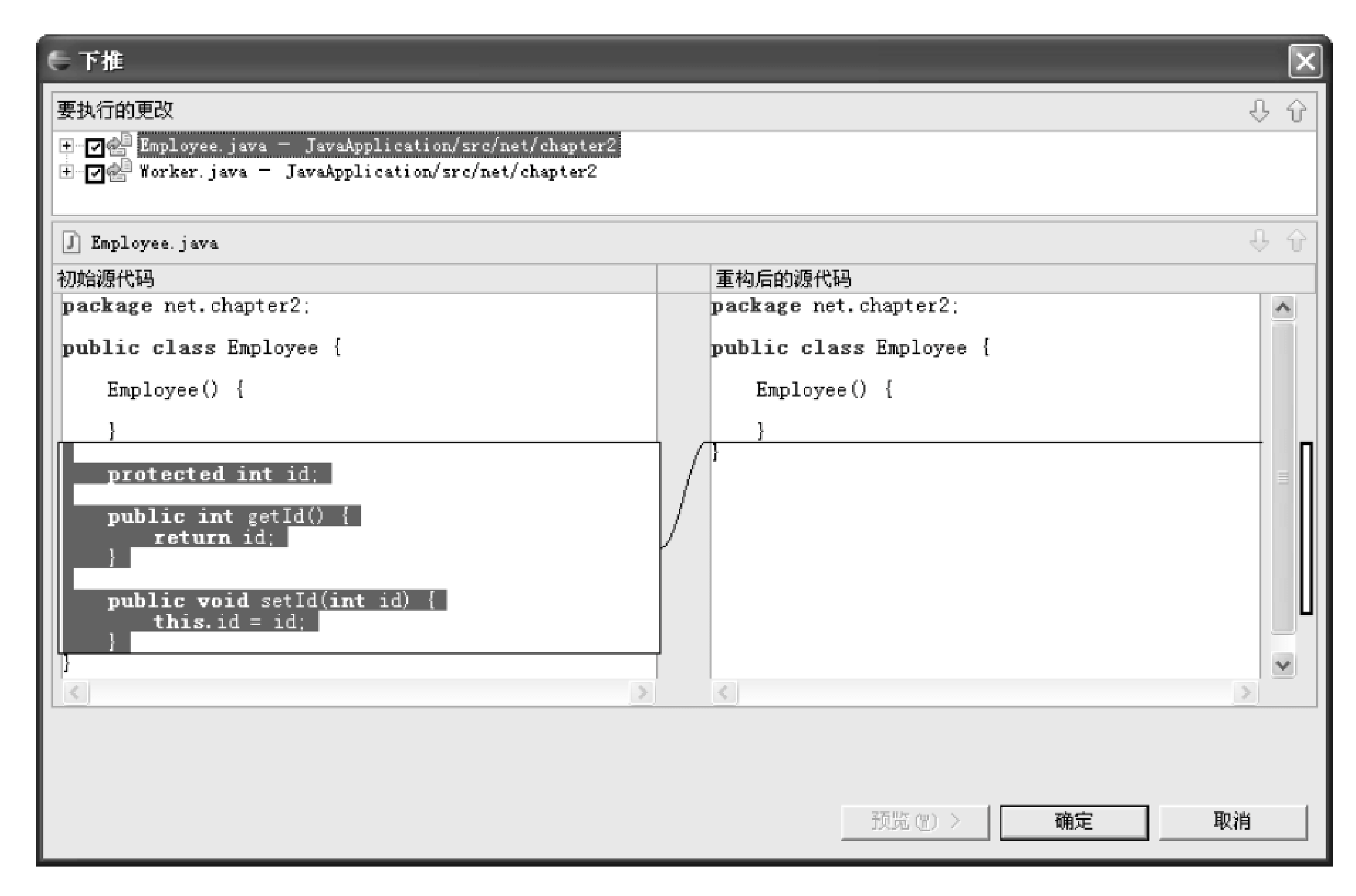


图 2-14　Employee.java 的改变

（4）在【要执行的更改】中单击“Worker.java”，预览 Worker.java 文件变化，如图 2-15 所示。

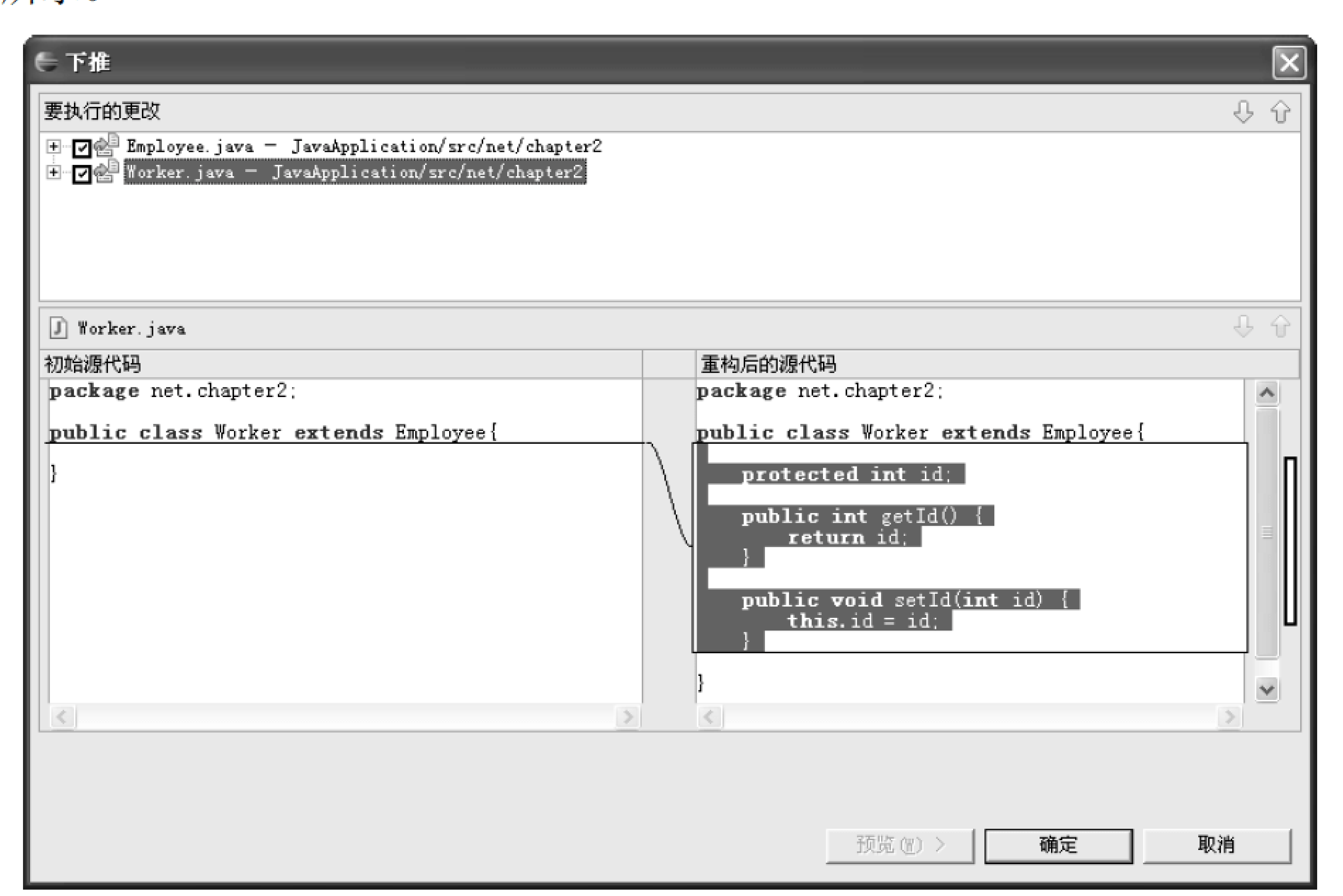


图 2-15　Worker.java 的改变

（5）单击【确定】按钮，完成重构。id 属性及其 getter/setter 又回到了 Worker.java 文件。

2.8　内联实例

“内联”重构可用于方法、静态 final 属性、局部变量，将它们用实际的代码代替。

跟我做

（1）为了说明“内联”重构功能，在 Util.java 中增加一个新的 statSalary 方法，用于统计员工薪水的等级。代码如下：

```
    public void statSalary(){
        //调用 countSalary 方法，统计 3 号员工的薪水等级，countSalary 方法的实现请参考
2.3 节
        int salary=countSalary(3);
        if(salary>=10000){
            System.out.println("High Salary!");        //当薪水多于 10000 时为高薪
        }else{
            System.out.println("Low Salary!");         //当薪水少于 10000 时为低薪
        }
    }
```

（2）双击方法名“countSalary”，使其被选中，单击【重构】菜单，选择【内联】命令，弹出【内联方法】对话框，如图 2-16 所示。



图 2-16　countSalary 方法的内联

- 【所有调用】单选按钮是指将所有 countSalary 方法的调用都进行内联。
- 【仅所选择的调用】单选按钮是指只将 Util.java 中本次选择的 countSalary 方法内联。

（3）单击【预览】按钮，statSalary 方法的变化如图 2-17 所示。

（4）单击【确定】按钮，完成重构。重构后的 statSalary 方法代码如下：

```
    public void statSalary(){
        int wage = 3000;            //工资
        int bonus = 1500;           //奖金
        int subsidy = 1000;         //津贴
        int salary=wage + bonus + subsidy;
        if(salary>=10000){
            System.out.println("High Salary!");
        }else{
            System.out.println("Low Salary!");
        }
    }
```



图 2-17　statSalary 方法的改变

2.9　抽取方法实例

“抽取方法”重构可创建一个包含当前所选择的语句或表达式的新方法，并将选择替换为对新方法的引用。可以使用编辑菜单中的扩大选择至以获取有效的选择范围。此功能对于清理冗长、杂乱或过于复杂的方法是很有用的。

跟我做

（1）为了说明“抽取方法”重构功能，将 2.8 节中内联的代码重新抽取成方法。用鼠标选中以下代码，选中部分变成蓝色。

```
int wage = 3000;//工资
int bonus = 1500;//奖金
int subsidy = 1000;//津贴
int salary=wage + bonus + subsidy;
```

（2）单击【重构】菜单，选择【抽取方法】命令，弹出【抽取方法】对话框，如图 2-18 所示。

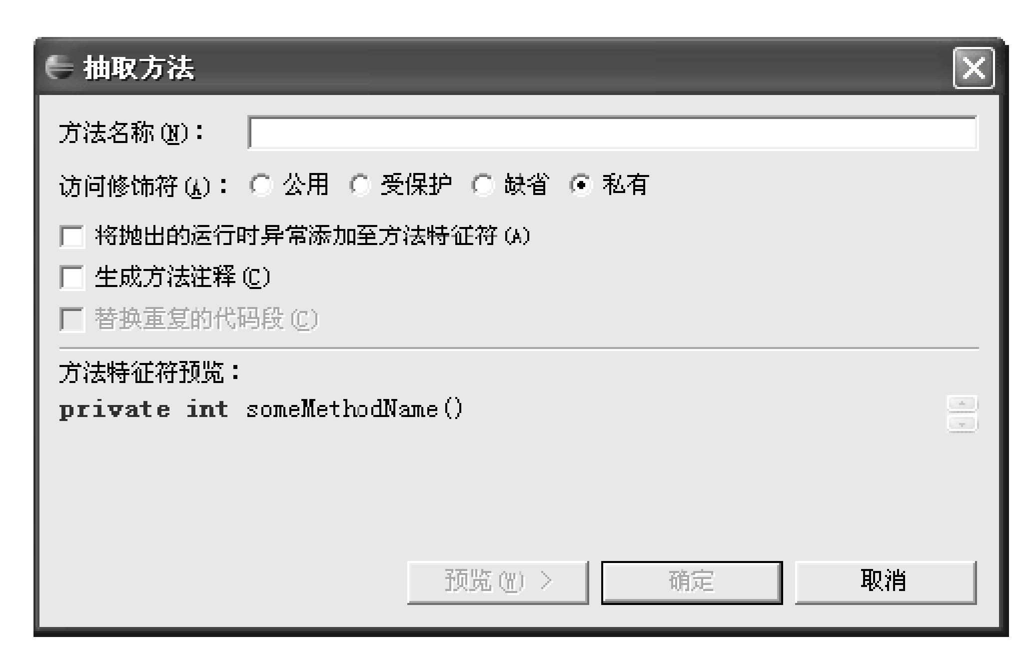


图 2-18　countSalary 方法的抽取

（3）在【方法名称】文本框中输入“countSalary”，单击【预览】按钮。程序的变

化如图 2-19 所示。

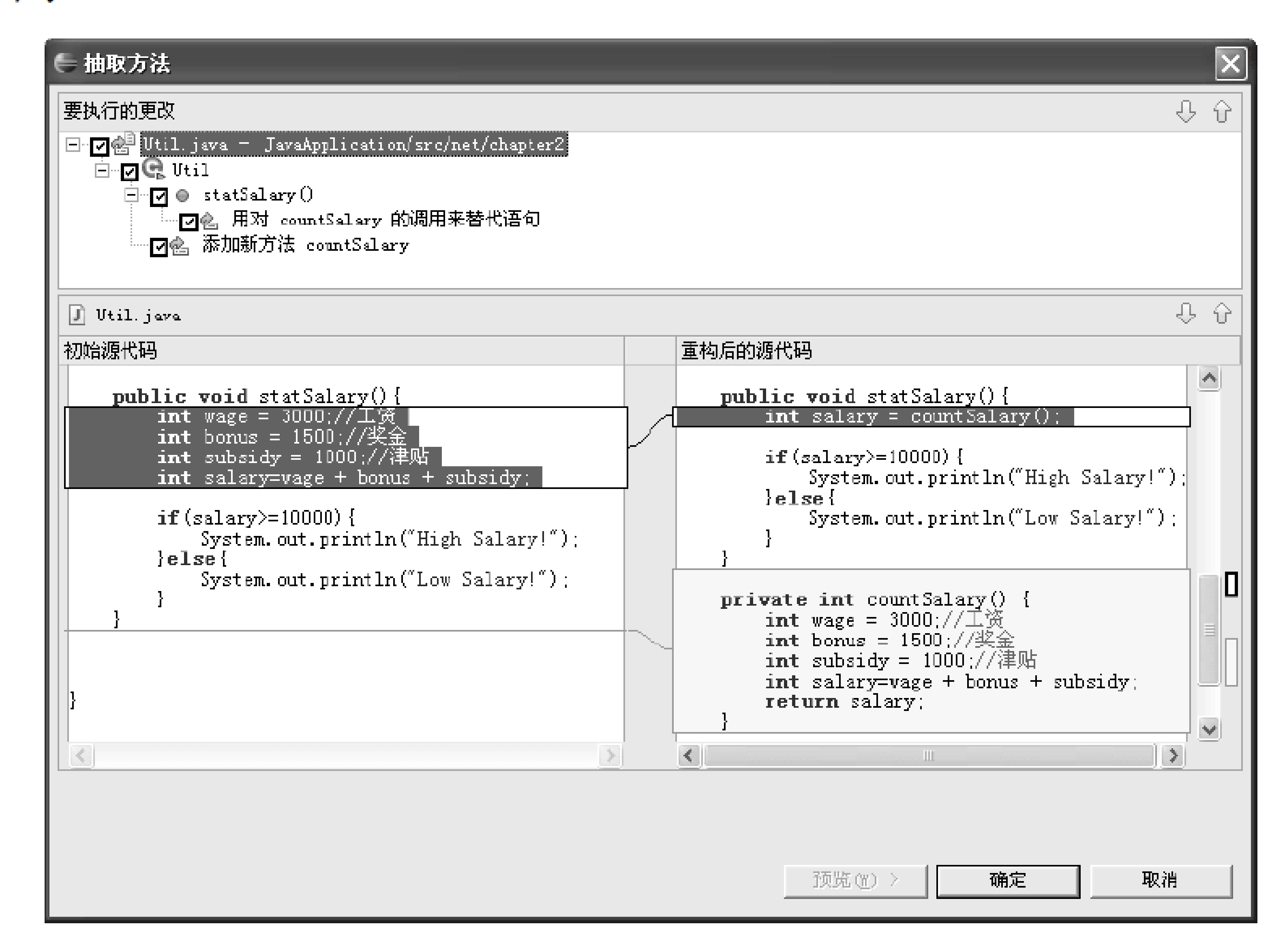


图 2-19　抽取方法前后的改变

（4）单击【确定】按钮，完成重构。

2.10　抽取常量实例

“抽取常量”重构可将所选表达式创建静态终态字段并替换字段引用，并且可以选择重写同一表达式的其他出现位置。该重构可用于静态终态字段和解析为静态终态字段的文本选择。

跟我做

（1）在 statSalary 方法中，为了复用向控制台输出的字符串“High Salary!”和“Low Salary!”，可将它们抽取成静态终态常量。用鼠标选中“High Salary!”字符串（包括两个引号）。

（2）单击【重构】菜单，选择【抽取常量】命令，弹出【抽取常量】对话框，如图 2-20 所示。



图 2-20　将字符串抽取成常量

（3）在【常量名】文本框中输入“HIGH_SALARY”，在【访问修饰符】中选中【公用】单选按钮，选中【将所有出现了所选表达式的地方都替换为对常量的引用】复选框，单击【预览】按钮。程序的变化如图 2-21 所示。

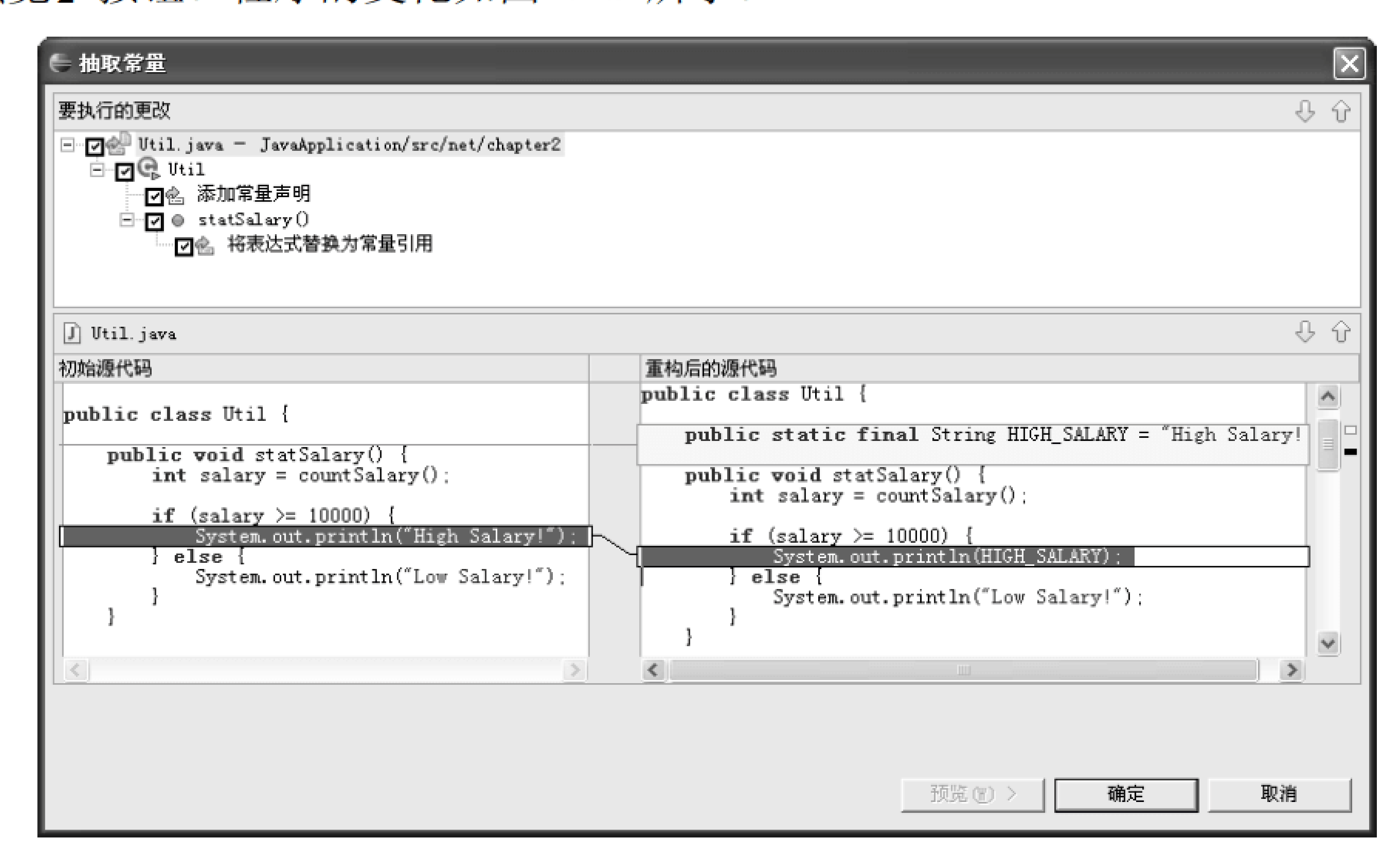


图 2-21　抽取常量前后的改变

（4）单击【确定】按钮，完成重构。

（5）继续将“Low Salary!”抽取成常量。重构后的代码如下：

```
public static final String LOW_SALARY = "Low Salary!"; //新抽取的常量 LOW_SALARY
public static final String HIGH_SALARY = "High Salary!"; //新抽取的常量 HIGH_SALARY

public void statSalary() {
        int salary = countSalary();
        if (salary >= 10000) {
                System.out.println(HIGH_SALARY);          //字符串已改变成常量
        } else {
                System.out.println(LOW_SALARY);           //字符串已改变成常量
        }
}
```

2.11　引入工厂实例

“引入工厂”重构可创建一个新的工厂方法，该方法将调用选择的构造函数并返回创建的对象。对该构造函数的所有引用都将被替换为对新工厂方法的调用。

跟我做

（1）Worker.java 有两个构造方法，代码如下：

```
//不带参数的构造方法
public Worker() {
```

```
    }

    //有两个参数的构造方法
    public Worker(int id, String name) {
        this.id = id;
        this.name = name;
    }
```

（2）将这两个构造方法分别改变成工厂模式。双击第一个构造方法的方法名，使其呈选中状态。单击【重构】菜单，选择【引入工厂】命令，弹出【引入工厂】对话框，如图 2-22 所示。



图 2-22　引入工厂模式

（3）【工厂方法名】文本框中自动出现“createWorker”字样，作为新引入的工厂方法的名称；【工厂类】文本框中自动出现构造函数所在类的包和名称。选中【使构造函数是私有的】复选框，单击【预览】按钮。程序的变化如图 2-23 所示。

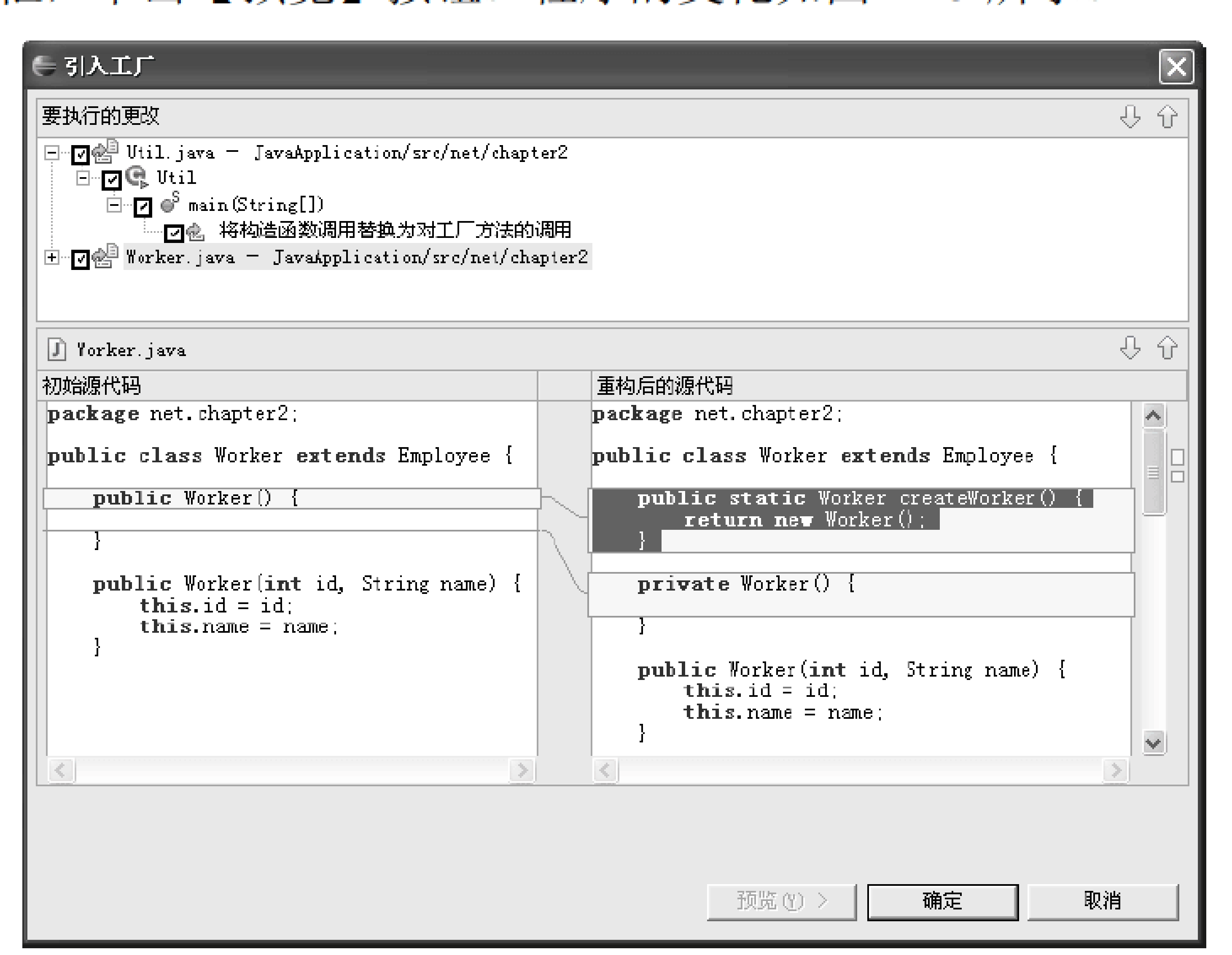


图 2-23　Worker()构造函数的变化

（4）单击【确定】按钮，完成重构。代码如下：

```
    public static Worker createWorker() {
        return new Worker();
    }
    private Worker() {
    }
```

构造函数的描述符已经变成了私有，增加了静态的 createWorker 方法，其返回值是 Worker 对象。当需要创建 Worker 对象时，可调用工厂方法完成。代码如下：

```
Worker worker = Worker.createWorker();
```

其他类将不能直接调用 Worker 类的私有构造函数，必须通过工厂方法实例化对象。

（5）继续将第二个构造方法重构，重构后的代码如下：

```
public static Worker createWorker(int id, String name) {
    return new Worker(id, name);
}

private Worker(int id, String name) {
    this.id = id;
    this.name = name;
}
```

第 3 章　Eclipse 插件使用实例

Eclipse 的价值是它为创建可扩展的集成开发环境提供了一个开放源码平台。这个平台允许任何人构建与环境和其他工具无缝集成的工具。它是一个成熟的、精心设计的以及可扩展的体系结构。工具与 Eclipse 无缝集成的关键是插件。除了小型的运行是内核之外，Eclipse 中的所有东西都是插件。从这个角度来讲，所有功能部件都是以同等的方式创建的。但是，某些插件比其他插件更重要些，例如 Workbench 和 Workspace 是 Eclipse 平台的两个必备插件，它们提供了大多数插件使用的扩展点。本章将通过实例介绍 Eclipse 常用插件，这些插件在编程过程中会带来极大的方便。

3.1　使用 XMLBuddy 编写 XML 文件

XMLBuddy 由 Bocaloco Software 开发，是一款编辑开发 XML（Extensible Markup Language，可扩展标记语言）文档的插件。它可免费下载，适用于 Windows、MacOS X、Linux 和 Solaris 操作系统。它为 Eclipse 增添了 XML 编辑能力，其中包括对用户可配置的语法着色、DTD 驱动的代码辅助、验证以及同步的提纲视图。XMLBuddy 还为 Workspace 添加 XML 透视图，为 XML 文档和 DTD 添加新的项目模板。

跟我做

1．下载 XMLBuddy

XMLBuddy 的主页网址为 http://www.xmlbuddy.com。注意要下载的是 XMLBuddy，而不是 XMLBuddy Pro。XMLBuddy Pro 提供了更强大的功能，但是需要付费。XMLBuddy 最新版本为 2.0.72，这个版本支持 Eclipse 3.1。下载 xmlbuddy_2.0.72.zip 包到本地，接下来安装 XMLBuddy 到 Eclipse。

2．安装 XMLBuddy

解压 xmlbuddy_2.0.72.zip，然后将 com.objfac.xmleditor_2.0.72 目录复制到%Eclipse%\plugins 目录下即可。启动 Eclipse，XMLBuddy 插件安装完成。需要注意的是，如果是 Windows NT 或 Windows 2000 操作系统，Eclipse 将不能自动检测 plugins 文件夹的改变，安装后使用 -clean 命令行参数重新启动 Eclipse，XMLBuddy 即可成功安装。

可以通过 Eclipse 插件详细信息查询功能，查看插件是否安装成功。单击【帮助】菜单，选择【关于 Eclipse SDK】命令，弹出【关于 Eclipse SDK】对话框。单击【插件详细信息】按钮，弹出【关于 Eclipse SDK 插件】对话框。这个列表包括了 Eclipse 所有插件信息，XMLBuddy 也在其中，如图 3-1 所示。



图 3-1　Eclipse SDK 插件详细信息

3．配置 XMLBuddy

在使用 XMLBuddy 之前，需要对其进行配置。步骤如下：

（1）单击【窗口】菜单，选择【首选项】命令，弹出【首选项】对话框。

（2）单击左侧列表中的【XMLBuddy】选项，在右侧面板中出现“XMLBuddy”配置信息，如图 3-2 所示。

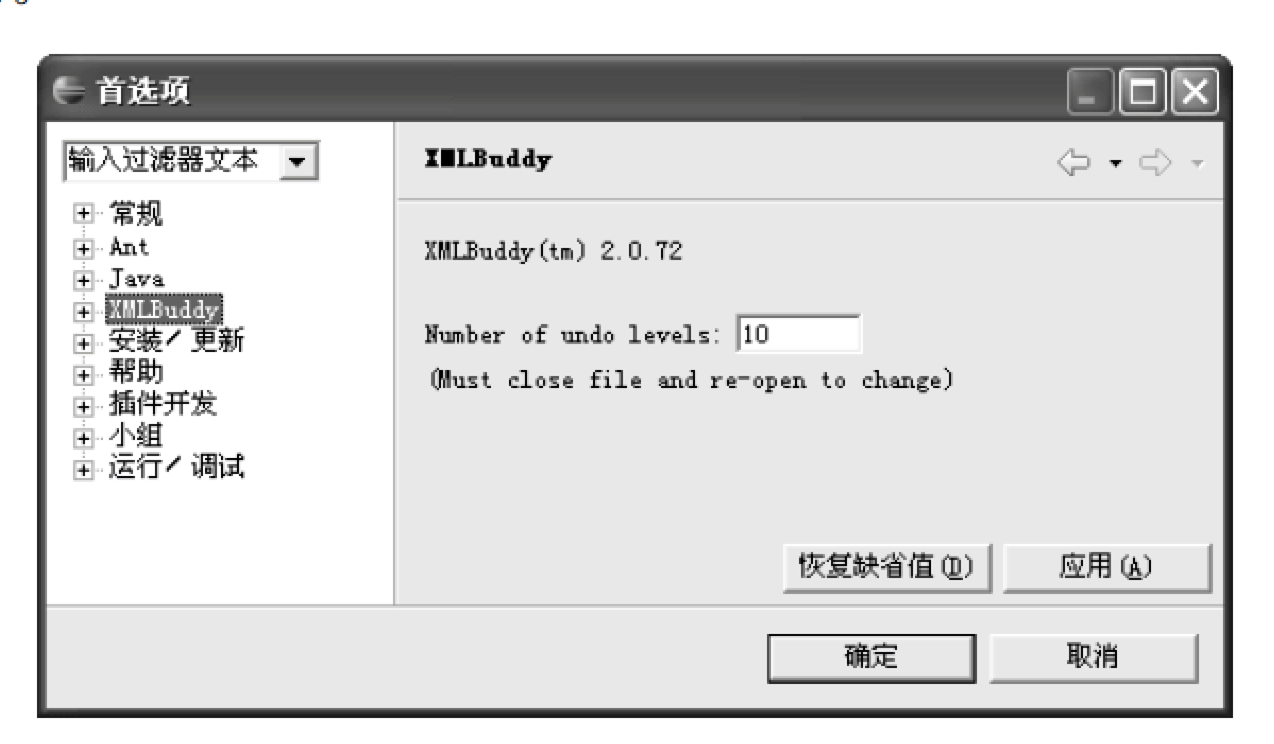


图 3-2　XMLBuddy 配置信息

（3）该面板可以设置编辑 XML 文件时“撤销”功能的深度级别，可在文本框中输入一个 10～100000 的数字。单击【应用】按钮，保存设置。

（4）单击左侧列表【XMLBuddy】选项前面的“+”号，展开更多的配置项。单击【Content Assist】选项，配置内容助手功能，如图 3-3 所示。

右侧面板中：

第一个复选框为使自动助手功能可用，延时时间为 0.5 秒。

第二个复选框为使自动完成功能可用。

第三个复选框为自动插入结束标签。

第四个复选框为自动删除多余的结束标签。

第五个复选框为在“</”之后自动填充结束标签名称。

全部选中 5 个复选框，单击【应用】按钮，保存设置。



图 3-3　内容助手配置

（5）单击左侧列表中的【Encoding】选项，配置 XML 文件编码格式，如图 3-4 所示。

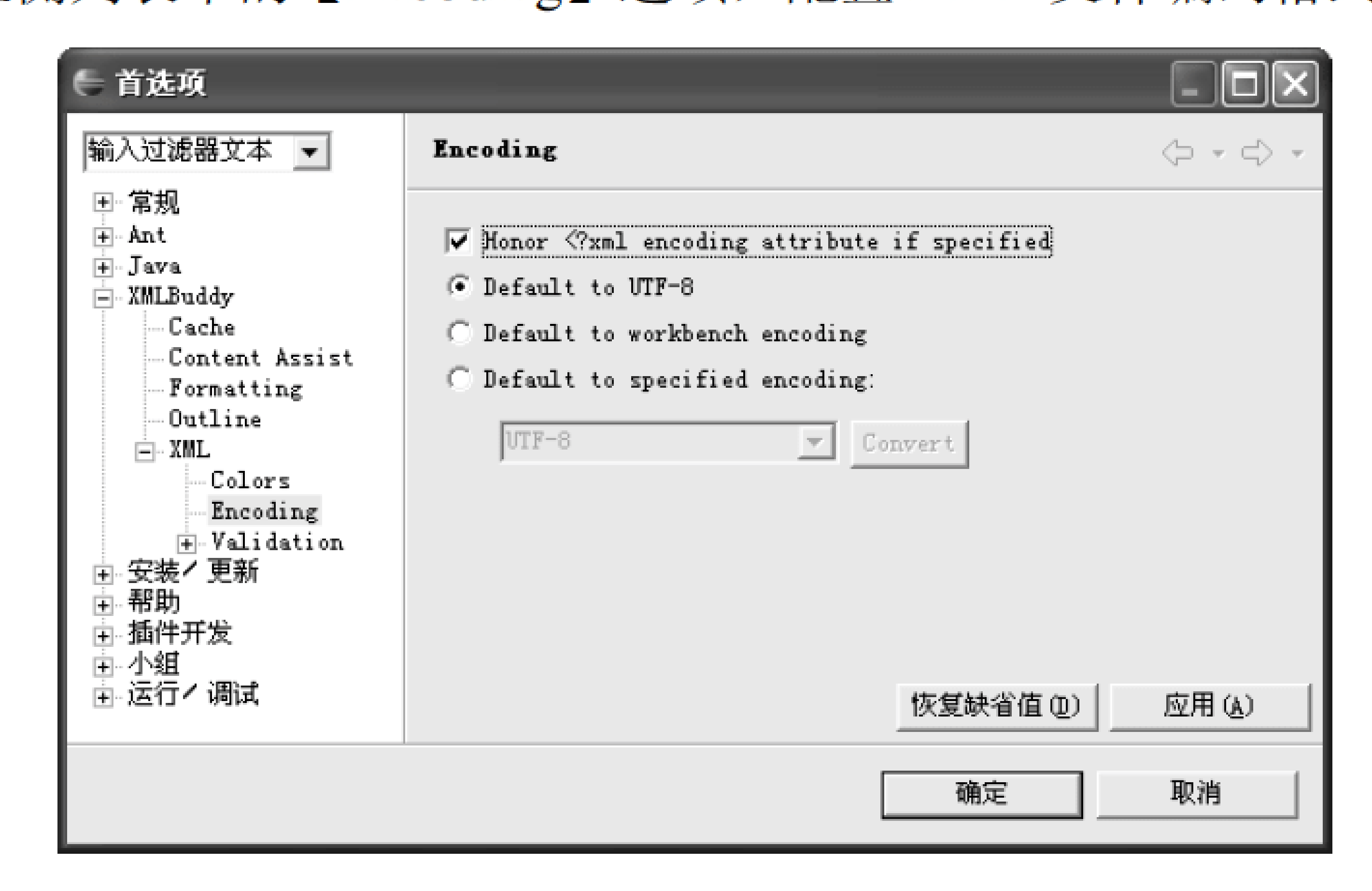


图 3-4　编码格式配置

在右侧面板中：

复选框为如果“<?xml”标签的 encoding 属性被指定，则使用此属性作为 XML 文件的编码格式。

第一个单选按钮指定 XML 文件默认编码格式为“UTF-8”。

第二个单选按钮指定 XML 文件默认编码格式与工作台相同。中文操作系统下，工作台编码格式为“GBK”。

第三个单选按钮指定具体编码格式作为 XML 文件默认的编码格式。

选中复选框和第一个单选按钮，单击【应用】按钮，保存设置。

其他的 XMLBuddy 配置选项均保持默认值，单击【确定】按钮，保存设置并退出对话框。

4．使用 XMLBuddy

本实例将创建一个 DTD 文件和一个 XML 文件，用于存储用户的注册信息。用户注册信息包括用户名、登录 ID、密码、电子邮件、电话和地址。其中，地址是可选项，其他选项均为必填项。

（1）创建 DTD 文件。单击工具栏中的 图标，弹出【New DTD】对话框，如图 3-5 所示。



图 3-5　创建 DTD 文件

（2）在【New DTD】文本框中输入“Users”，单击【下一步】按钮，出现【Encoding Declaration】界面，如图 3-6 所示。

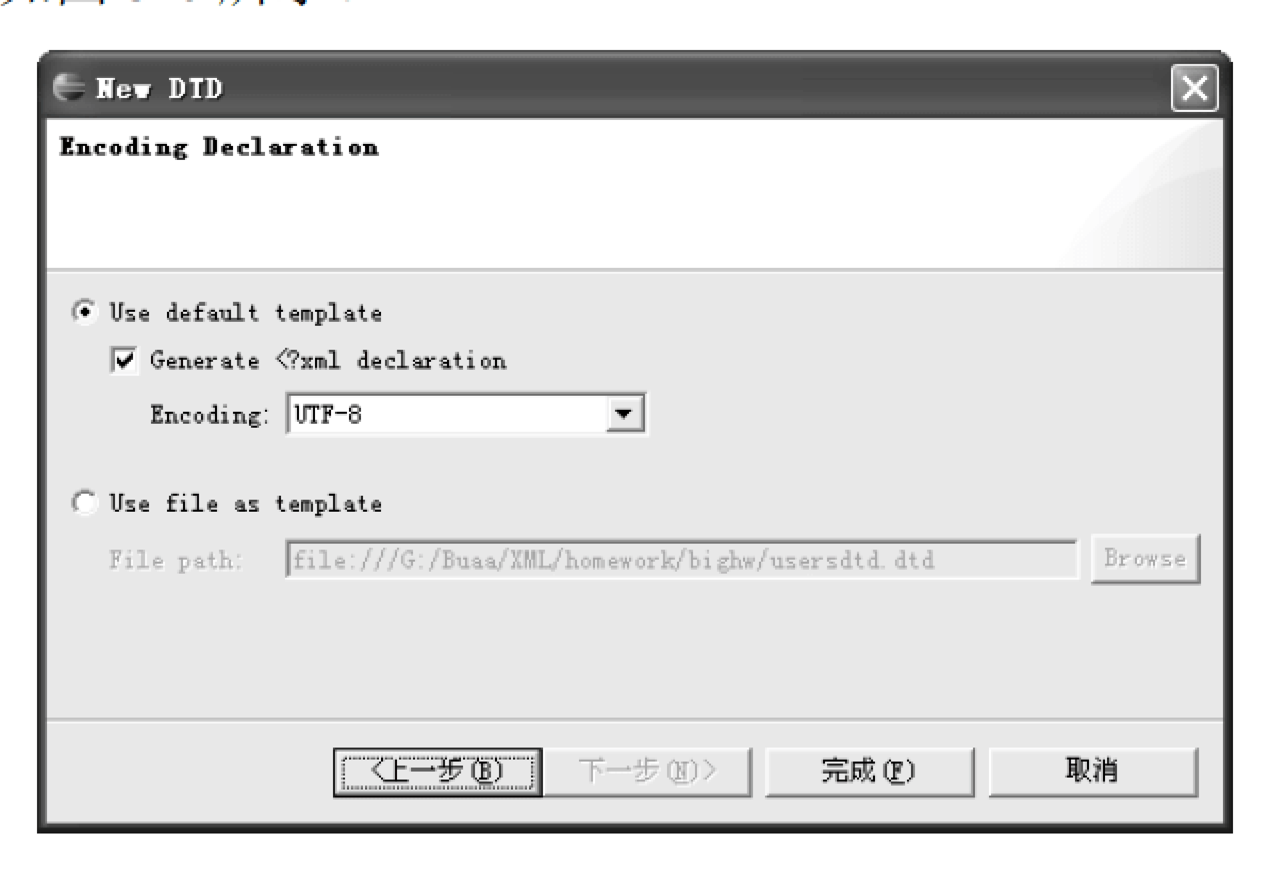


图 3-6　选择编码格式

（3）保留默认设置，单击【完成】按钮，编辑器自动打开 Users.dtd 文件，编写 Users.dtd 文件。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT Users (User*)><!--定义根节点，可存在 0 或多个 User 节点-->
<!ELEMENT User (UserName,LoginID,Password,Email,PhoneNumber,Address?)><!--定义 User 节点-->
<!ELEMENT UserName (#PCDATA)><!--定义用户名节点-->
<!ELEMENT LoginID (#PCDATA)><!--定义登录 ID 节点-->
<!ELEMENT Password (#PCDATA)><!--定义密码节点-->
```

```
<!ELEMENT Email (#PCDATA)><!--定义电子邮件节点-->
<!ELEMENT PhoneNumber (#PCDATA)><!--定义电话节点-->
<!ELEMENT Address (#PCDATA)><!--定义地址节点-->
```

（4）创建 XML 文件。单击工具栏中的图标，弹出【New XML Document】对话框，如图 3-7 所示。

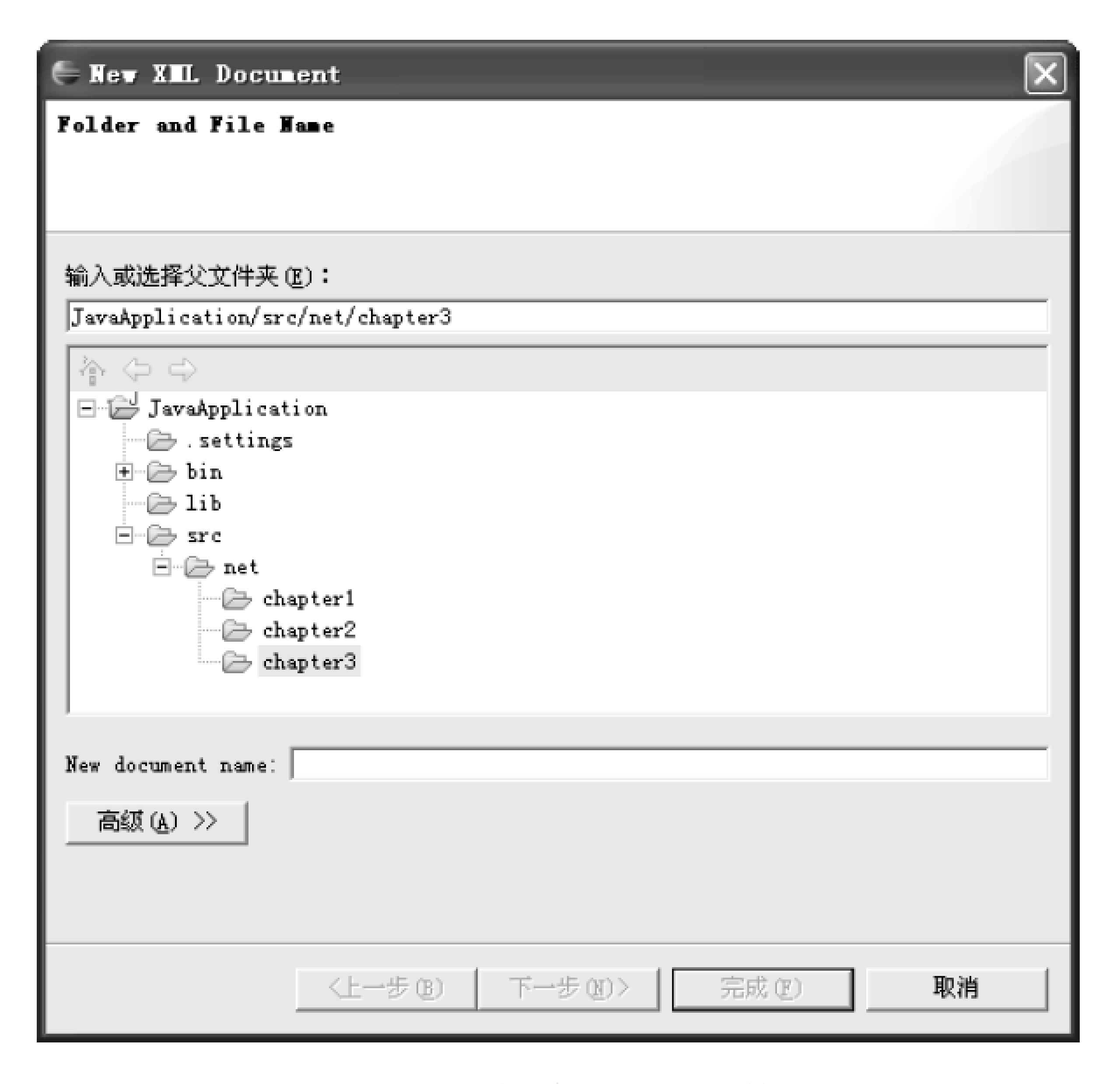


图 3-7　创建 XML 文件

（5）在【New document name】文本框中输入“Users”，单击【下一步】按钮，出现【Template】界面，如图 3-8 所示。

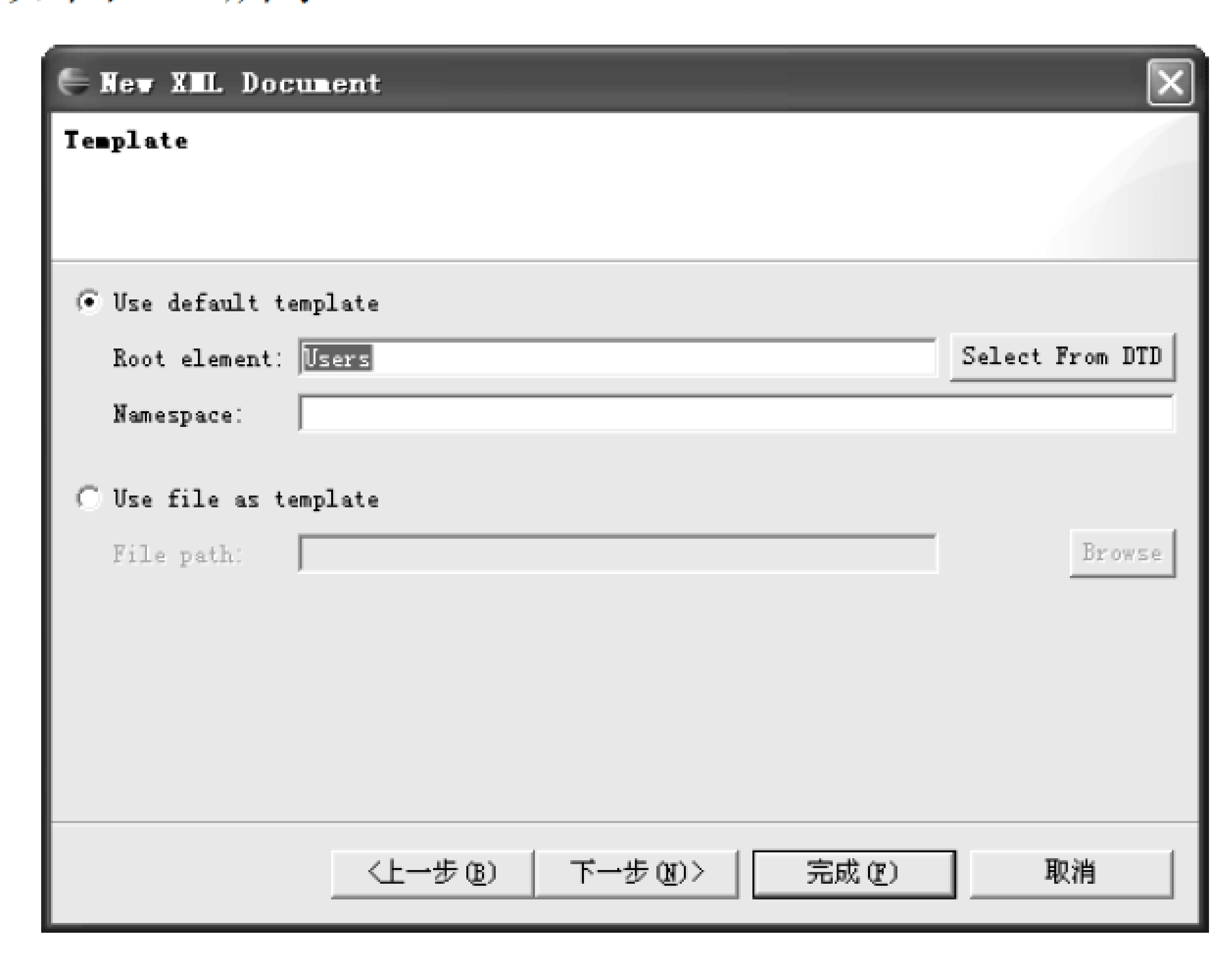


图 3-8　选择模板

（6）单击【Select From DTD】按钮，弹出【Select Root】对话框，如图 3-9 所示。

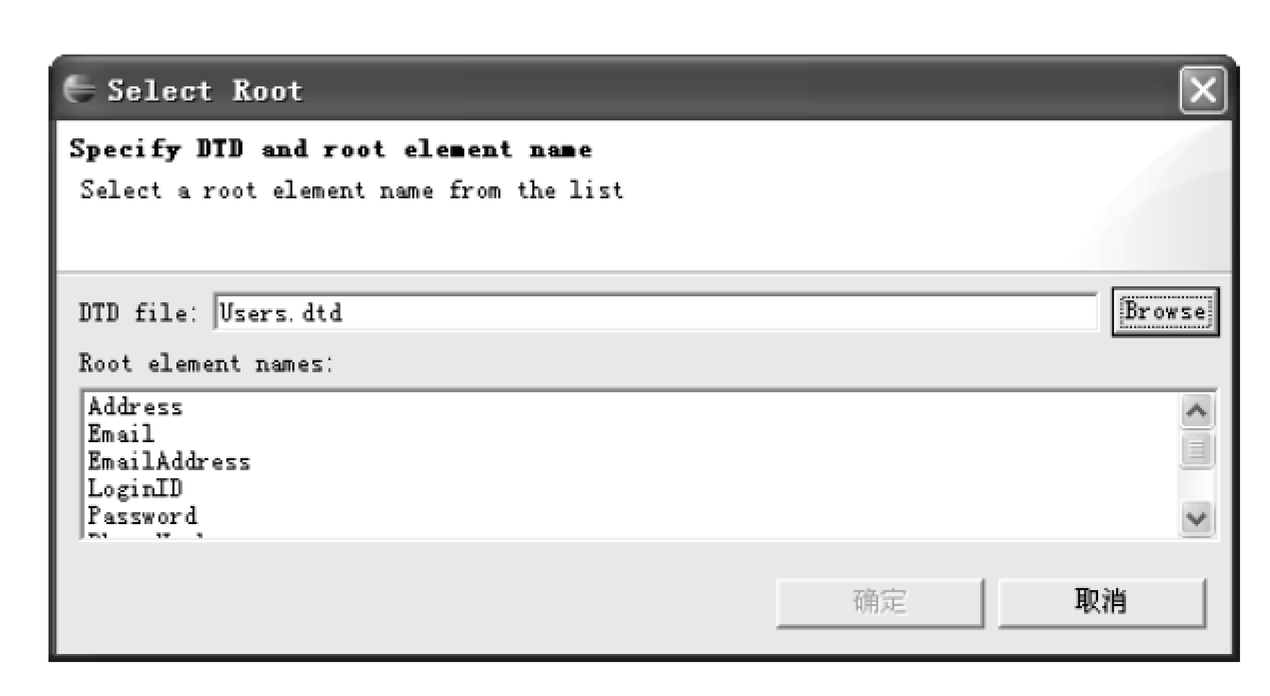


图 3-9　选择根节点

（7）单击【Browse】按钮，选择刚才创建的 Users.dtd 文件，在【Root element names】列表框中选中【users】选项，单击【确定】按钮，关闭对话框并回到【Template】对话框。单击【下一步】按钮，出现【DTD Or Schema】对话框，如图 3-10 所示。

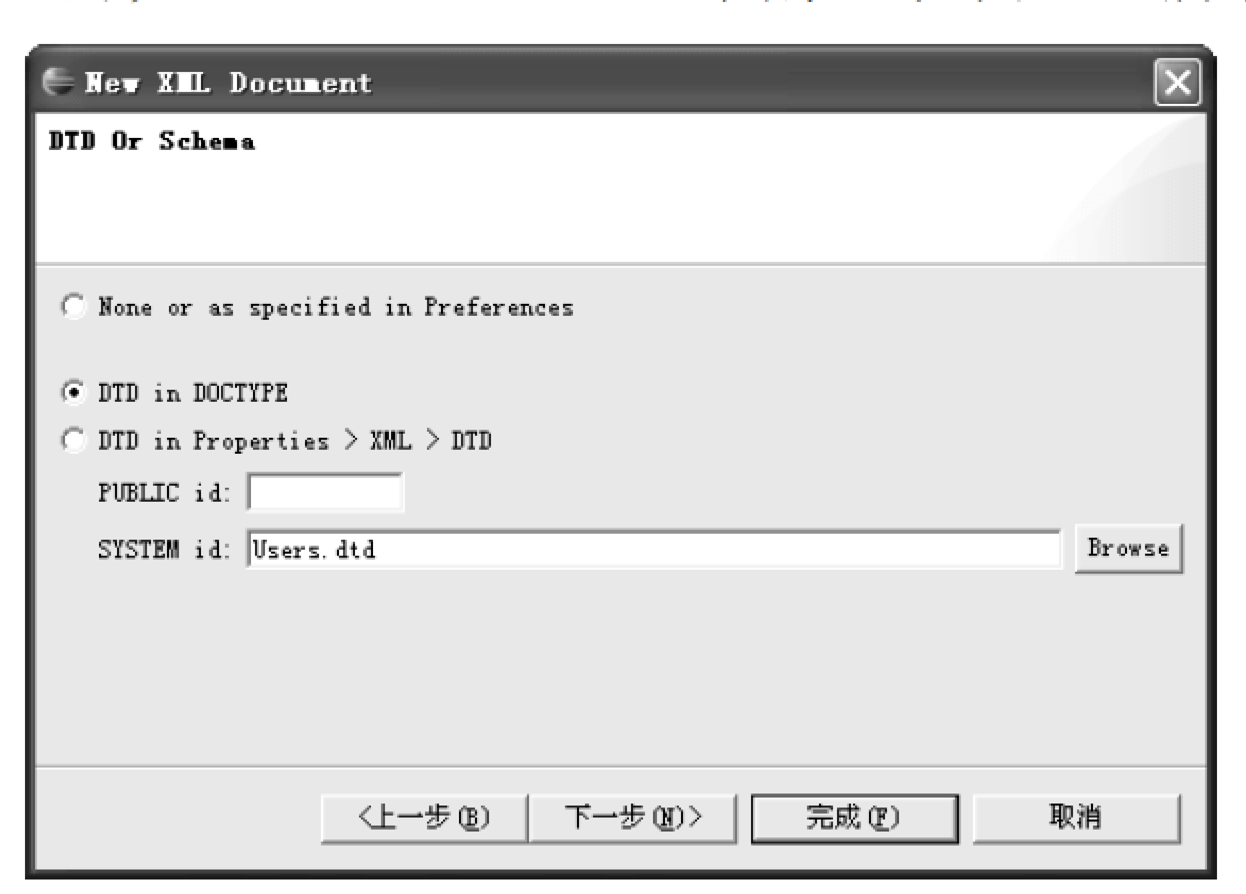


图 3-10　选择 DTD

（8）单击【下一步】按钮，出现【Declarations】对话框，如图 3-11 所示。

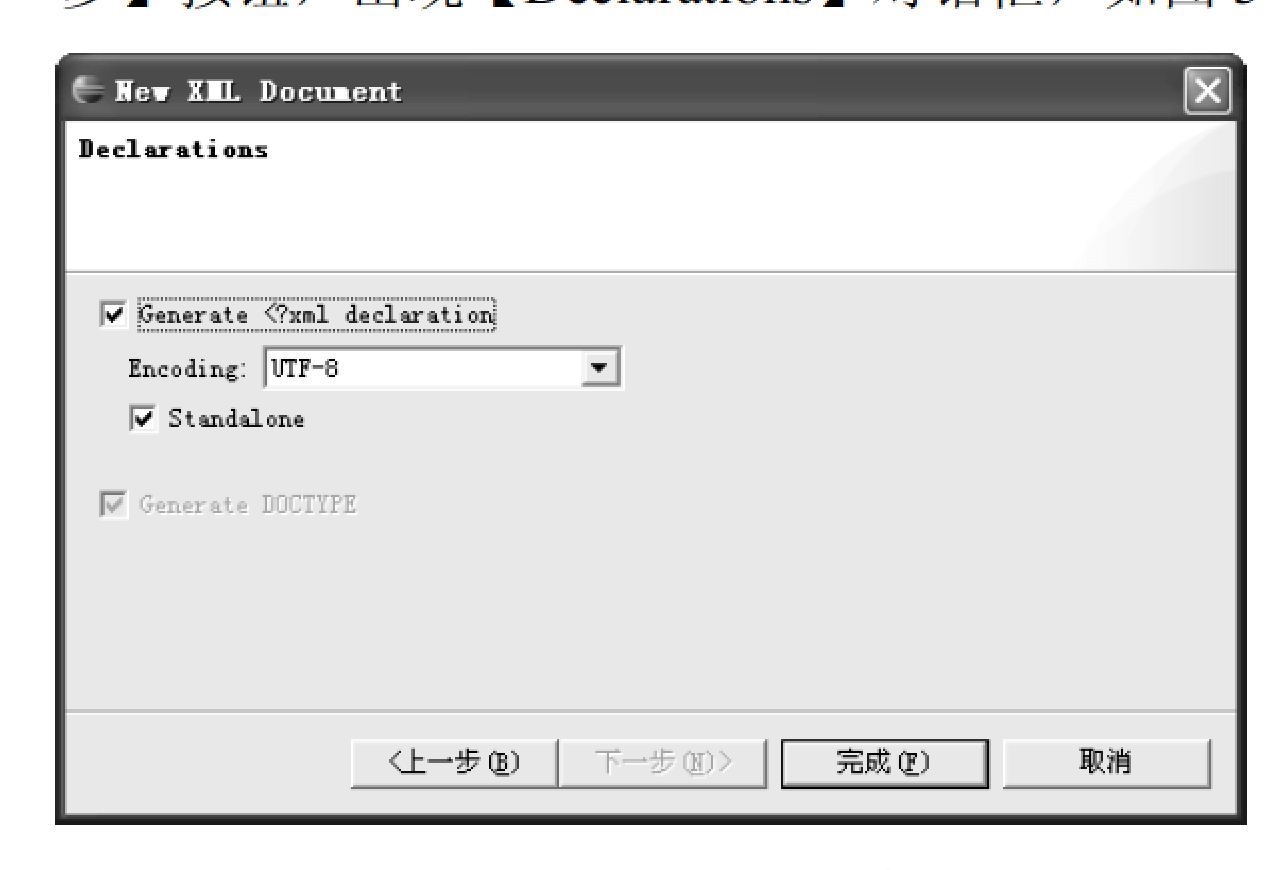


图 3-11　配置 XML 定义

（9）取消选中【Standalone】复选框，单击【完成】按钮，编辑器自动打开 Users.xml 文件，编写 Users. xml 文件。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Users SYSTEM "Users.dtd">
<Users>
    <User>
        <UserName>zhangsan</UserName>
        <LoginID>zhangsan</LoginID>
        <Password>zhangsan</Password>
        <Email>zhangsan@eyou.com</Email>
        <PhoneNumber>12345</PhoneNumber>
        <Address>beijing</Address>
    </User>
</Users>
```

（10）单击【XML】菜单，可对 XML 文件进行 DTD 验证、格式化的操作。

3.2 使用 Bytecode Outline 直接查看字节码

Bytecode Outline 插件可以直接查看字节码，把正在编辑 Java 的文件或者 class 文件直接显示出其相应的字节码。可以进行两个 Java 文件的字节码比较或者两个 class 文件的字节码比较或者一个 Java 文件与一个 class 文件的字节码比较。

跟我做

1．下载并安装 Bytecode Outline

Bytecode Outline 插件的下载和安装将通过更新管理的形式来进行。

（1）启动 Eclipse 后，单击【帮助】菜单，依次选择【软件更新】|【查找并安装】命令，弹出【安装/更新】对话框，如图 3-12 所示。

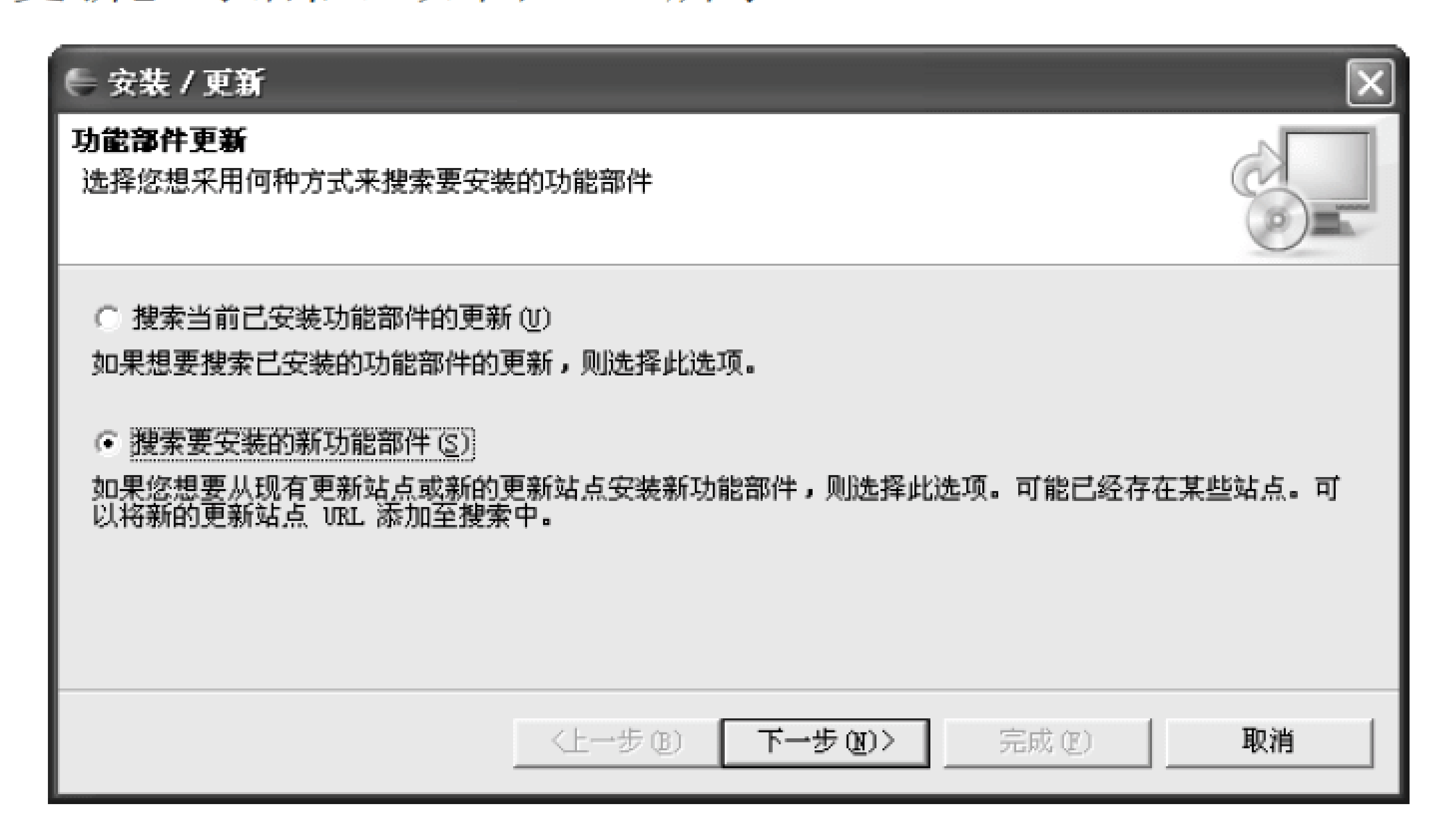


图 3-12 【安装/更新】对话框

（2）选中【搜索要安装的新功能部件】单选按钮，单击【下一步】按钮，弹出【安装】对话框，如图 3-13 所示。



图 3-13　安装插件

（3）单击【新建远程站点】按钮，弹出【新建更新站点】对话框，如图 3-14 所示。



图 3-14　【新建更新站点】对话框

（4）在【名称】文本框中输入“Bytecode Outline”，在【URL】文本框中输入“http://download.forge.objectweb.org/eclipse-update/”，单击【确定】按钮，在【安装】对话框中的【要包括在搜索中的站点】列表中出现 Bytecode Outline 选项。

（5）选中【Bytecode Outline】选项，单击【完成】按钮，Eclipse 自动搜索当前站点可用的插件。搜索完成后，出现【更新】对话框，如图 3-15 所示。



图 3-15　【更新】对话框

（6）选中【选择要安装的功能部件】列表中所有选项，其中的 ASM Framework 是运行 Bytecode Outline 必需的框架。单击【下一步】按钮，出现【安装】对话框，如图 3-16 所示。

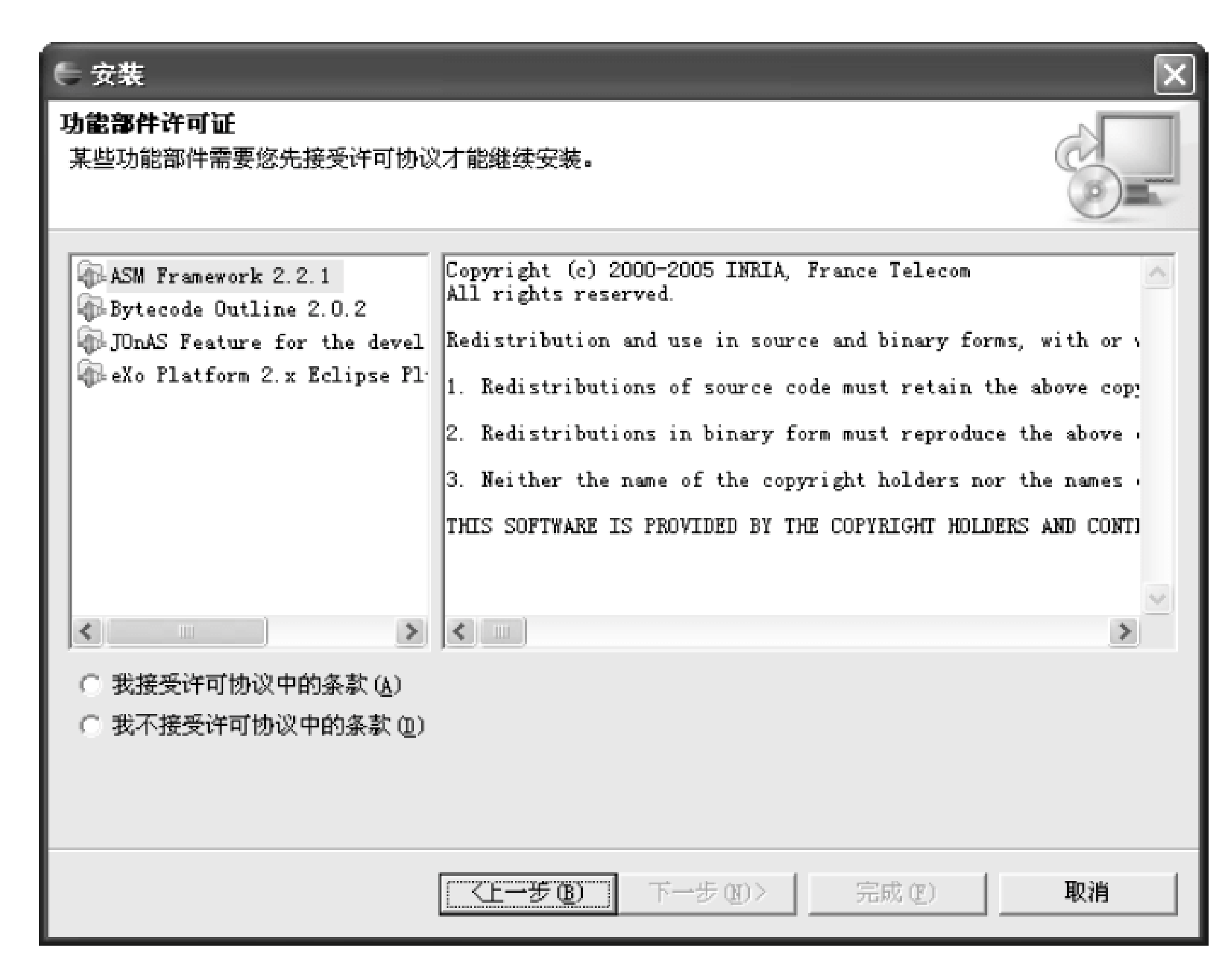


图 3-16　【安装】对话框

（7）选中【我接受许可协议中的条款】单选按钮，单击【下一步】按钮，出现安装配置确认对话框，如图 3-17 所示。



图 3-17　安装配置确认

（8）单击【完成】按钮，出现【更新管理器】对话框，正式下载 Bytecode Outline 插件（这个过程要持续几分钟，视网速而定），如图 3-18 所示。可以单击【在后台运行】按钮，在后台运行下载过程，不影响在 Eclipse 中进行工作。



图 3-18　下载更新 Bytecode Outline 插件

（9）下载完成后，出现【验证】对话框，如图 3-19 所示。



图 3-19　【验证】对话框

（10）单击【全部安装】按钮，安装已下载的插件，如图 3-20 所示。



图 3-20　安装插件的进度信息

（11）安装完成后，弹出对话框，要求重新启动 Eclipse。单击【是】按钮，重新启动 Eclipse，安装完成。

2．使用 Bytecode Outline

（1）为了演示说明 Bytecode Outline 插件的使用方法，需要准备一个接口和两个实现类。右击“src”文件夹，依次选择【新建】|【包】命令，弹出【新建 Java 包】对话框。在【名称】文本框中输入“net.chapter3”，单击【完成】按钮。

（2）右击“net.chapter3”包，依次选择【新建】|【接口】命令，弹出【新建 Java 接口】对话框，如图 3-21 所示。



图 3-21　【新建 Java 接口】对话框

（3）在【名称】文本框中输入“MyInterface”，单击【完成】按钮，为接口增加“sayHello”方法。完整的代码如下：

```
public interface MyInterface {
    public void sayHello();
}
```

该接口只有一个方法，执行了该接口的类要实现这个方法。

（4）右击“net.chapter3”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“Implementor1”，单击【完成】按钮。类 Implementor1 的代码如下：

```
public class Implementor1 implements MyInterface {
    public void sayHello() {
        System.out.println("Hello Implementor1!");
    }
}
```

（5）右击“net.chapter3”包，依次选择【新建】|【类】命令，弹出【新建 Java 类】对话框。在【名称】文本框中输入“Implementor2”，单击【完成】按钮。类 Implementor2 的代码如下：

```
public class Implementor2 implements MyInterface {
    public void sayHello() {
        System.out.println("Hello Implementor2!");
    }
}
```

（6）打开 Bytecode 和 Bytecode Reference 视图。单击【窗口】菜单，依次选择【显示视图】|【其他】命令，弹出【显示视图】对话框。单击列表中【Java】选项左边的“+”号，列出【Java】选项下可选的视图，选中 Bytecode 和 Bytecode Reference 视图后，单击【确定】

按钮如图 3-22 所示。



图 3-22　打开 Bytecode 和 Bytecode Reference 视图

（7）打开“MyInterface.java”文件，在【Bytecode】窗口中可以看到相应的字节码，如图 3-23 所示。

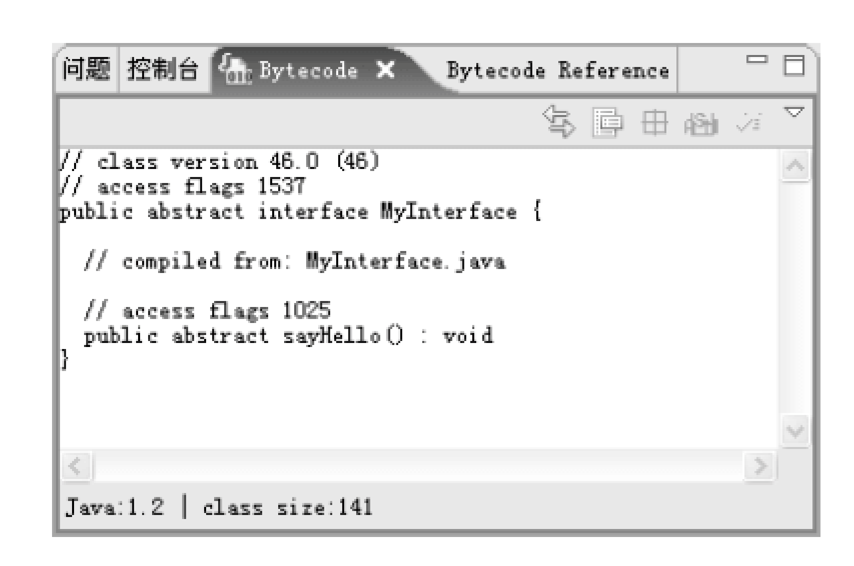


图 3-23　MyInterface.java 的字节码

在“Bytecode”视图的右上方有如下 5 个图标：

- ❑ 选中图标，当 Java 编辑器中当前的 Java 文件改变时，Bytecode 视图中的内容会跟着进行改变，方便查看。
- ❑ 选中图标，当 Java 编辑器中当前的 Java 文件定位在某个域或方法时，则仅显示域或方法的字节码，方便进行查看。当 Java 编辑器中定位在 sayHello 方法上时，Bytecode 视图仅显示这个方法的相关信息。
- ❑ 选中图标，会把相关的附加信息也显示，比如下面的 MyInterface，就会把 net/chapter3/包名都显示出来了，如图 3-24 所示。

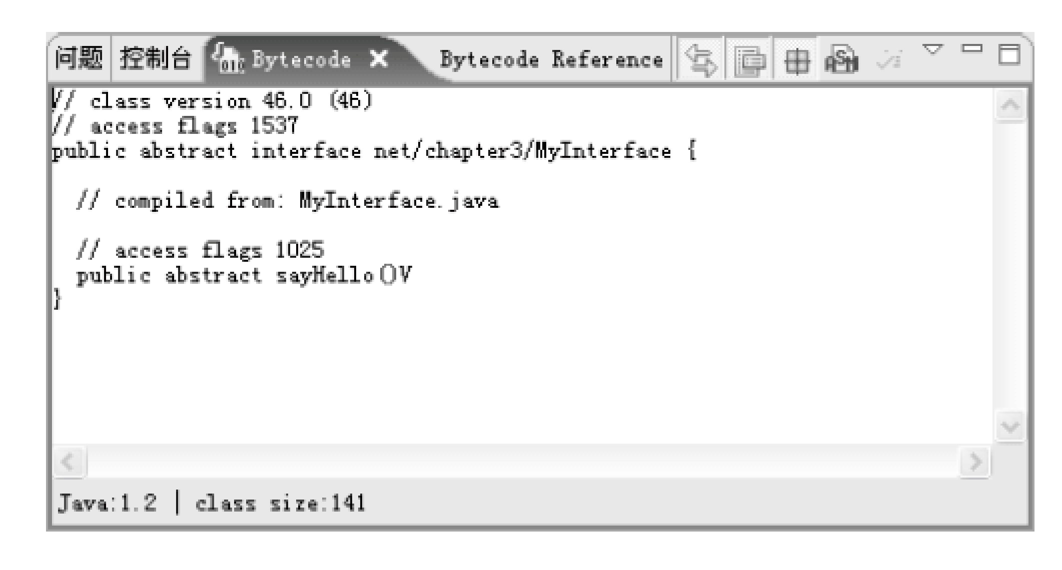


图 3-24　显示附加信息

❑ 选中图标时，可以在 true bytecode 和 ASMifier Java code 视图间进行切换。接口 MyInterface 的 ASMifier Java code 如图 3-25 所示。

```
package asm.net.chapter3;
import java.util.*;
import org.objectweb.asm.*;
import org.objectweb.asm.attrs.*;
public class MyInterfaceDump implements Opcodes {

public static byte[] dump () throws Exception {

ClassWriter cw = new ClassWriter(false);
FieldVisitor fv;
MethodVisitor mv;
AnnotationVisitor av0;

cw.visit(V1_2, ACC_PUBLIC + ACC_ABSTRACT + ACC_INTERFACE, "net/chapter3/MyInterface", null, "java/lang/Object", null);

cw.visitSource("MyInterface.java", null);

{
mv = cw.visitMethod(ACC_PUBLIC + ACC_ABSTRACT, "sayHello", "()V", null, null);
mv.visitEnd();
}
cw.visitEnd();

return cw.toByteArray();
}
}
Java:1.2 | class size:141
```

图 3-25　接口 MyInterface 的 ASMifier Java code

❑ 选中图标时，可以切换到字节码的指令集视图。当光标定位到不同的字节码时，就会同样定位到相应的指令上。同样，点击指令，相应地会定位到指令所对应的字节码。打开 Implementor1.java 文件，可以看到视图效果如图 3-26 所示。

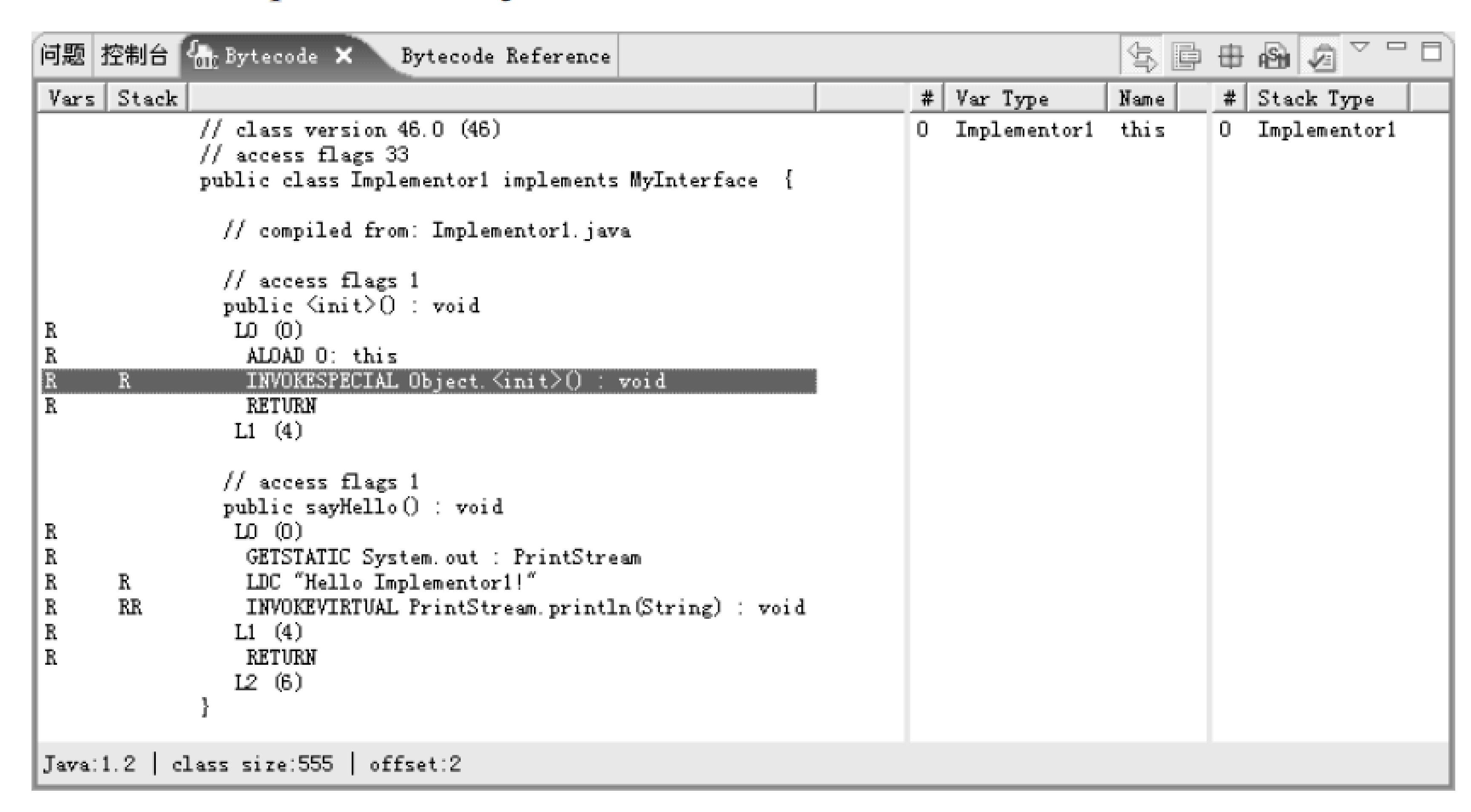


图 3-26　字节码和指令的对应

（8）比较两个 Java 文件的字节码。同时选中 Implementor1.java 和 Implementor2.java，右击并弹出快捷菜单，依次选择【比较对象】|【Each Other Bytecode】命令，比较两个文件的字节码，如图 3-27 所示。

也可以同时比较两个 class 文件的字节码或者比较一个 Java 文件和一个 class 文件的字节码。这里不再赘述，读者可自行练习。

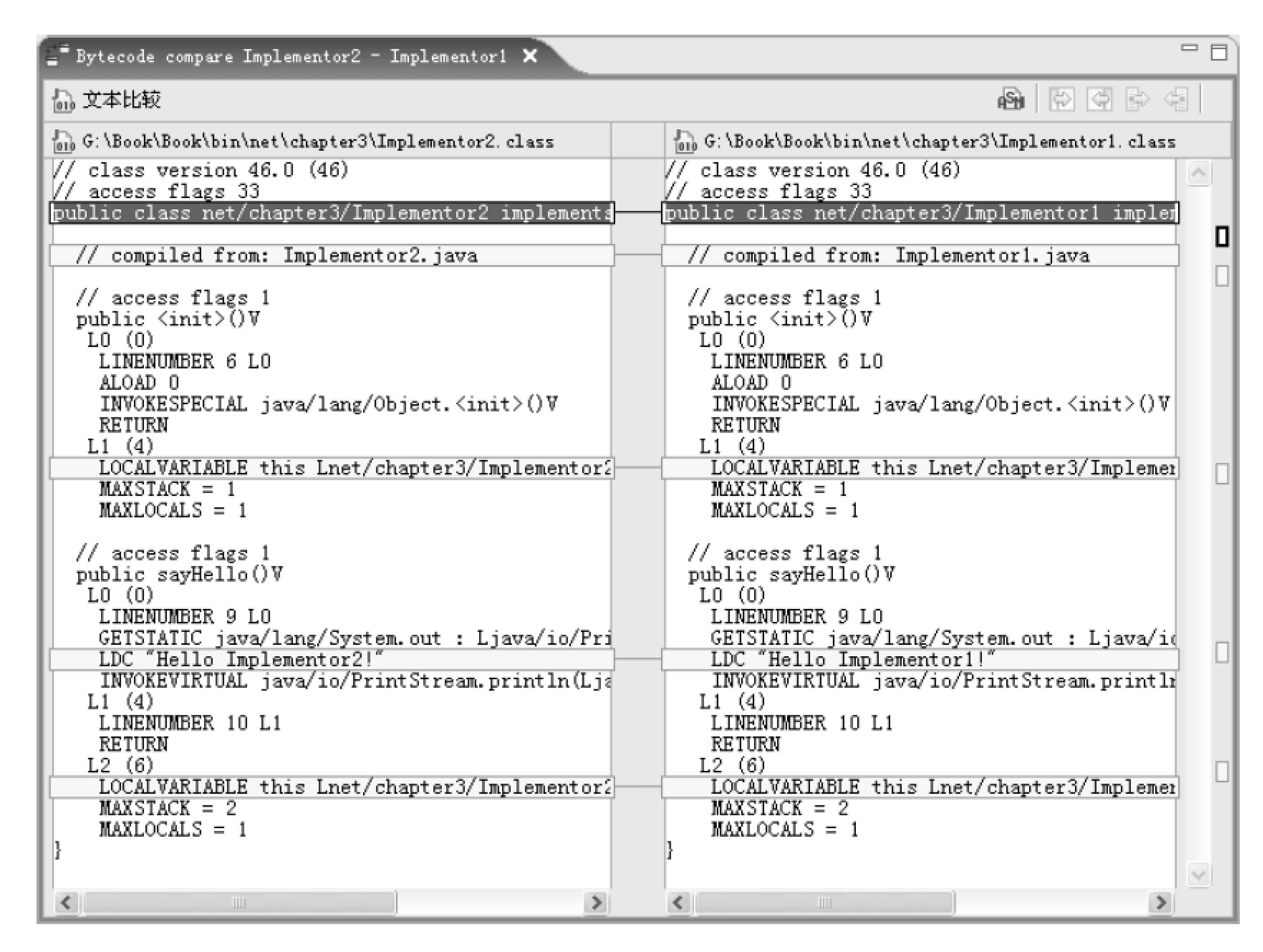


图 3-27　比较字节码

3.3　使用 Implementors 跟踪接口的实现类

Eclipse 提供了一个功能，即按住 Ctrl 键，单击某个类或方法，就可打开此类或者方法的具体实现代码。但是如果碰到接口时，只是到达接口而已，不能到达具体的实现类。Implementors 插件提供了跟踪接口实现类的功能，可以直接到达实现了接口的类的代码。

跟我做

1．下载 Implementors

Implementors 的主页网址：http://eclipse-tools.sourceforge.net/implementors。具体下载网址：http://superb-west.dl.sourceforge.net/sourceforge/eclipse-tools/dk.kamstruplinnet.implementors-0.0.15.zip。最新版本为 0.0.15，此版本支持 Eclipse 3.1。下载 dk.kamstruplinnet.implementors-0.0.15.zip 包到本地，接下来安装 Implementors 到 Eclipse。

2．安装 Implementors

安装 Implementors 的方式很简单，解开压缩包，将 features 文件夹中的内容复制到%Eclipse%\features 目录下，将 plugins 文件夹中的内容复制到%Eclipse%\plugins 目录下，重新启动 Eclipse，即可完成安装。

3．使用 Implementors

（1）为了演示说明 Implementors 插件的使用方法，继续使用 3.2 节创建的 MyInterface 接口、Implementor1 类和 Implementor2 类。再增加一个 Demo 类调用 MyInterface 接口提供

的方法。Demo 类的代码如下：

```
public class Demo {
    public static void main(String[] args) {
        MyInterface myInferface=new Implementor1();
        myInferface.sayHello();
    }
}
```

（2）使用打开接口功能。在编辑器的 Implementor1.java 文件中，双击选中 sayHello 方法并右击，在弹出的快捷菜单中选择【Open Interface】命令，在编辑器中自动打开 MyInterface.java 文件，如图 3-28 所示。

（3）使用打开执行器功能。在编辑器中打开 Demo.java 文件，按住 Ctrl 键并单击调用的 sayHello 方法。这时，在编辑器中只能打开 MyInterface.java 文件，不能自动打开具体的 MyInterface 接口的执行类。可以使用 Implementors 插件的功能直接打开 MyInterface 接口的执行类。



图 3-28　打开接口

（4）双击选中 sayHello 方法并右击，在弹出的快捷菜单中选择【Open Implementation】命令，如图 3-29 所示。

（5）选择【Open Implementation】命令后弹出【Select Implementations】对话框，列表中列出了所有执行了 MyInterface 接口的类，如图 3-30 所示。



图 3-29　打开执行器



图 3-30　选择执行 MyInterface 接口的类

（6）选中 Implementor1 类，单击【确定】按钮，在编辑器中自动打开 Implementor1.java 文件。

3.4　使用 CAP 进行代码分析

Code Analysis Plugin（CAP）是一款 Java 代码分析插件，可以分析 Java 程序的从属关系、包和类的依赖关系，有助于提高体系结构的封装性、重用性和可维护性。CAP 打开一个独立的透视图用不同的视图和图表来显示分析结果。

跟我做

1．下载和安装 Code Analysis Plugin

Code Analysis Plugin 的主页网址：http://cap.xore.de。这个插件是由德国人开发的，文

档也是德文的，英文文档不够全面。下载和安装可通过更新管理的形式来进行。更新网址：http://cap.xore.de/update。

2．使用 Code Analysis Plugin

（1）使用 CAP 分析 JavaApplication 项目。右击“JavaApplication”项目，选择【Show CA】命令，弹出【Select project to be analyzed】对话框，如图 3-31 所示。

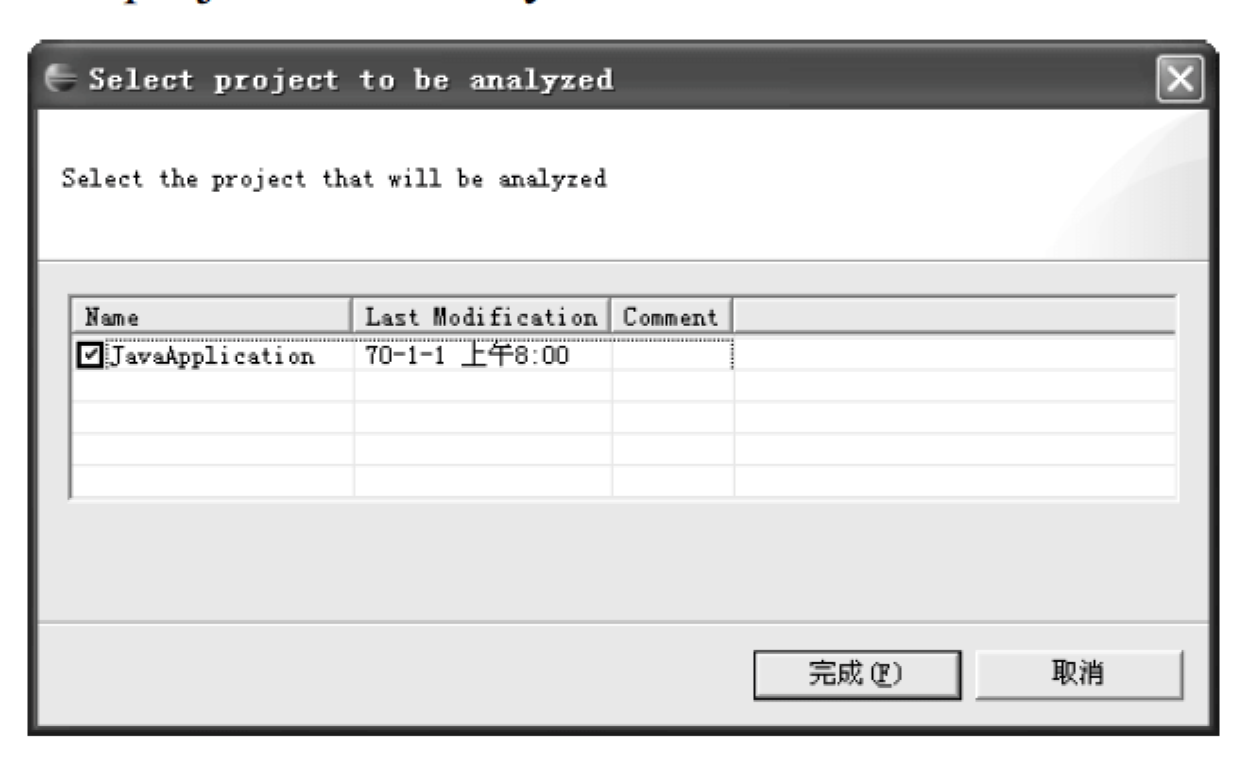


图 3-31　选择待分析的项目

（2）单击【完成】按钮，Eclipse 转换到 CAP 透视图。此透视图包含多个 CAP 视图，从多个方面分析包和类的关系，如图 3-32 所示。

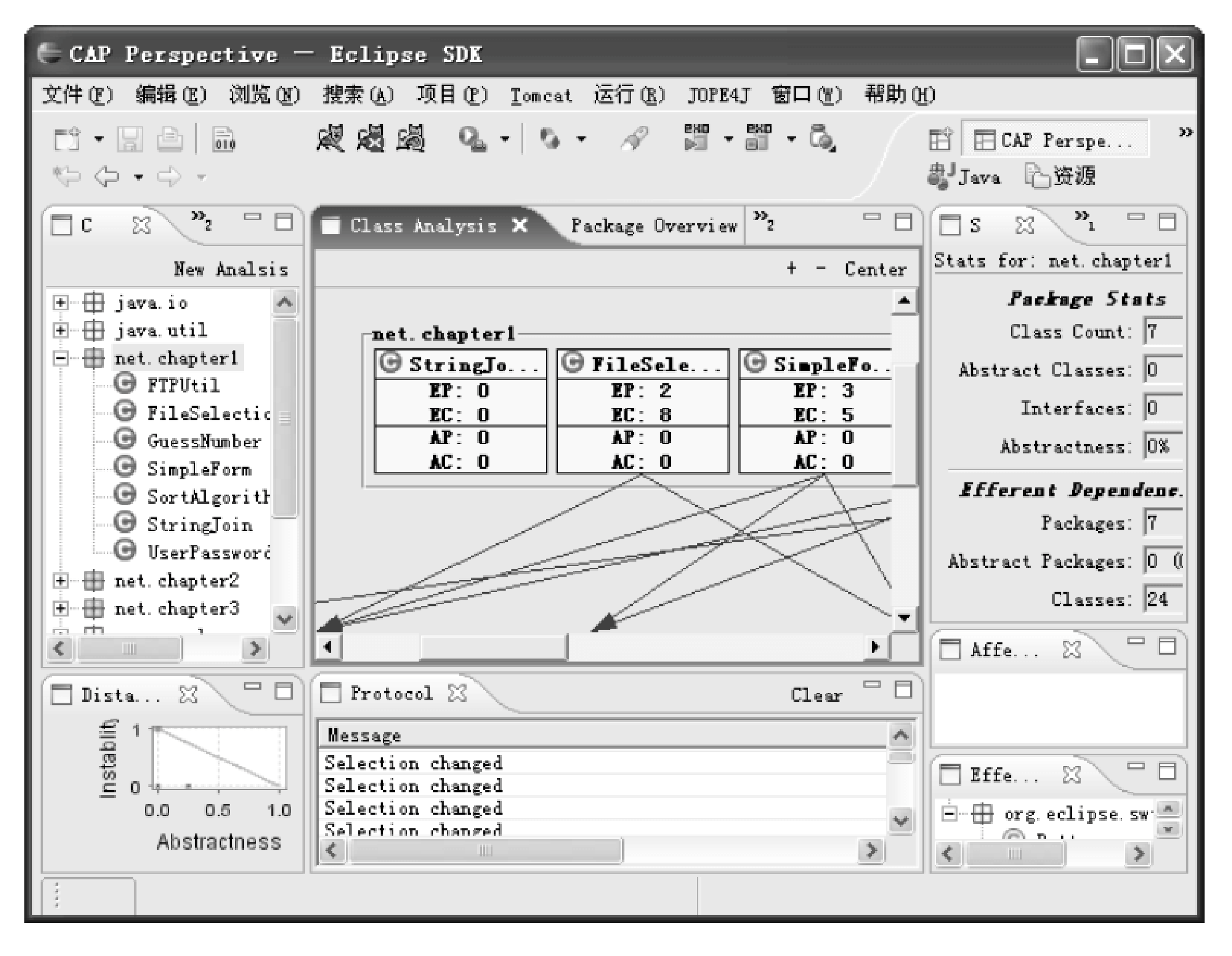


图 3-32　CAP 透视图

（3）在 CAP Package Explorer 视图中，可以看到所有与项目有关的包和类，包括项目中创建的包和类以及项目依赖的第三方的包和类，如图 3-33 所示。

（4）在 Class Analysis 视图中，可以看到项目中的类与引用的类的依赖关系。黄色的部分为项目中创建的类，如图 3-34 所示。

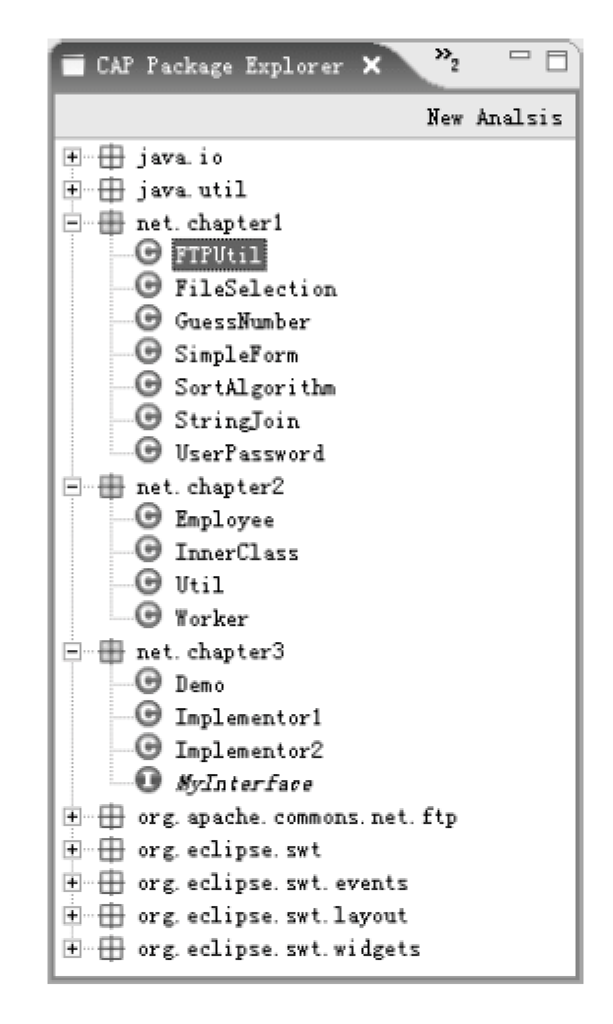


图 3-33　CAP Package Explorer 视图

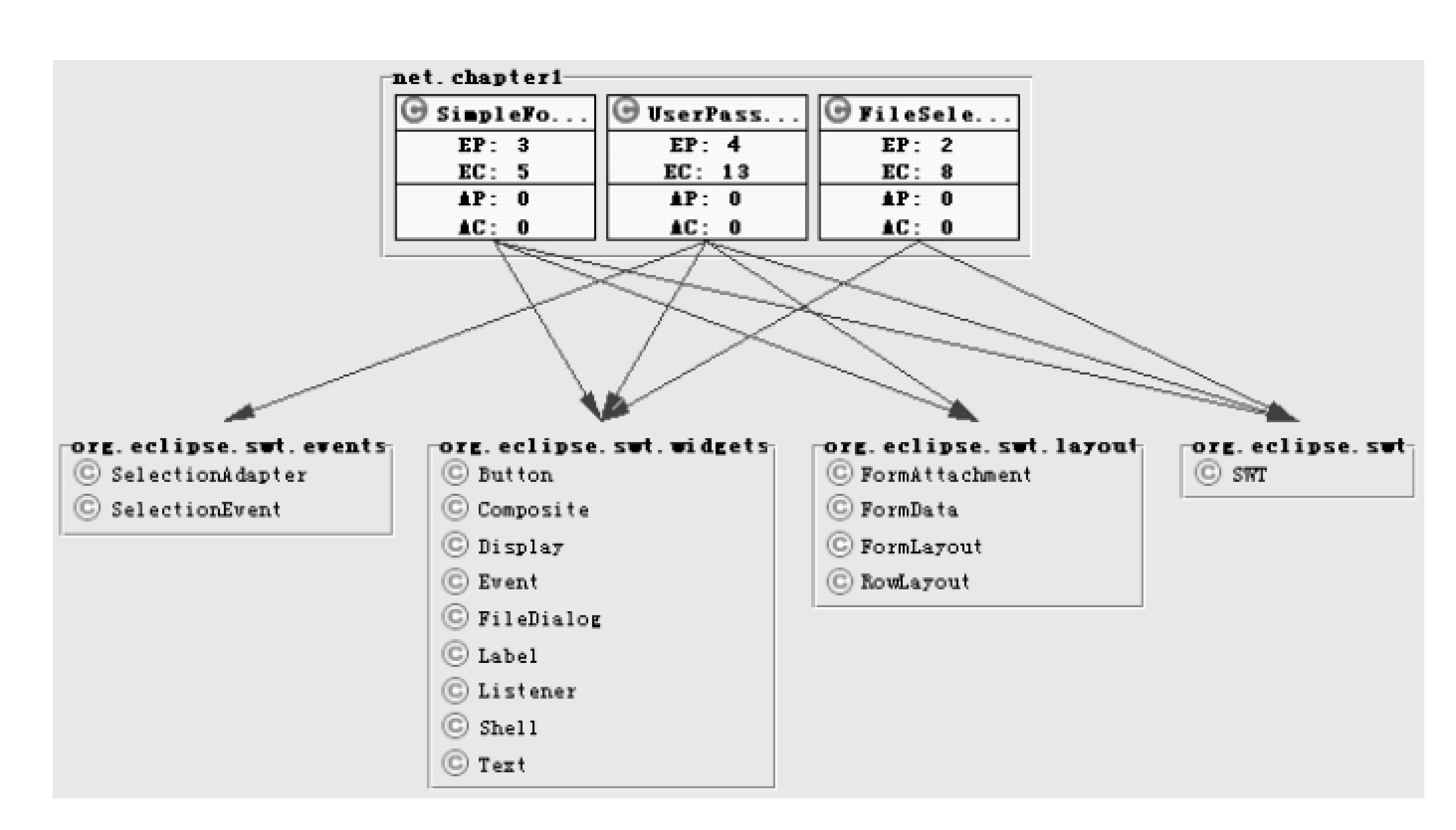


图 3-34　Class Analysis 视图

（5）在 Statistic 视图中，以包为单位统计分析类、接口、抽象类、引用的包和类、被引用包和类的数量，如图 3-35 所示。

（6）在 Protocol 视图中，记录在 CAP 透视图的操作日志，如图 3-36 所示。

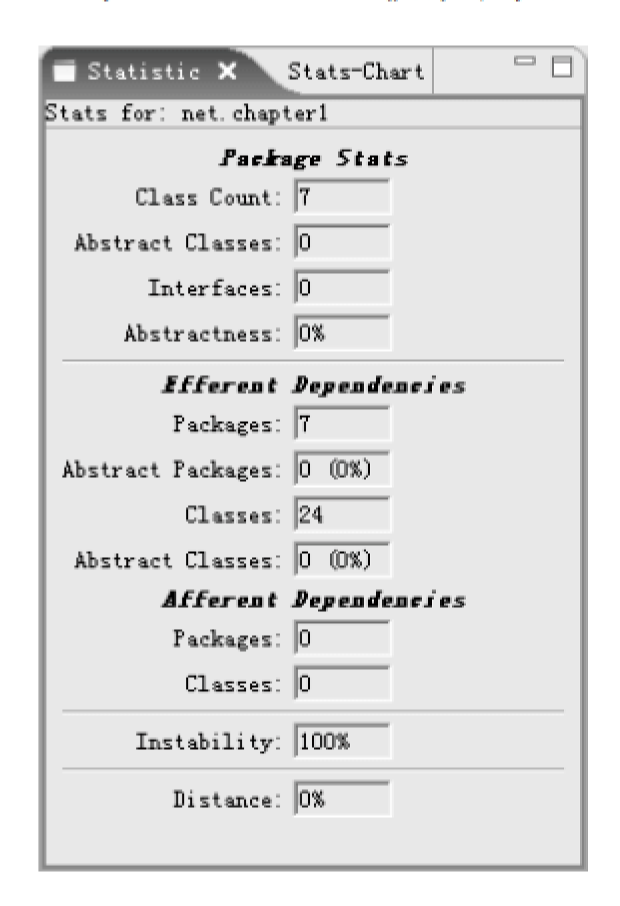


图 3-35　Statistic 视图

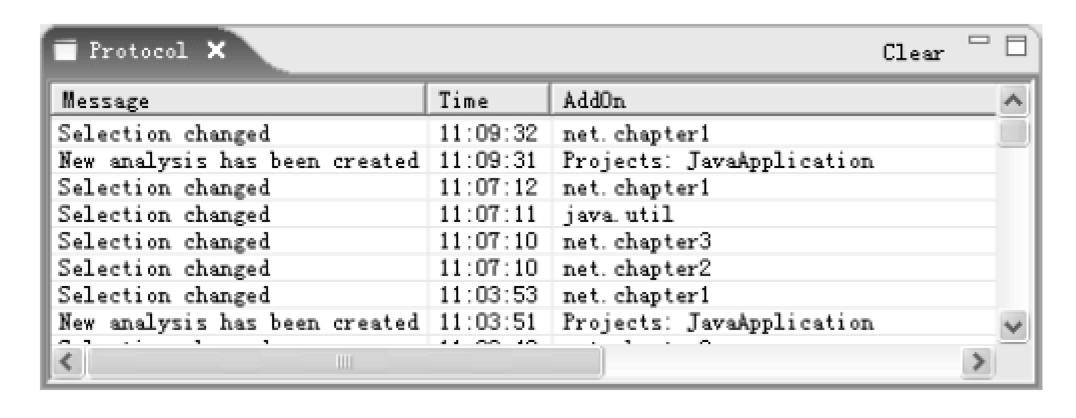


图 3-36　Protocol 视图

3.5　使用 Easy Explorer 快速查看文件夹

在 Eclipse IDE 环境中，如果想快速找到某个文件或文件夹所在的目录，并在操作系统的资源管理器中打开它，只能在资源管理器中一层一层地去查找其所在目录，这样比较麻烦。Easy Explorer 能够快速定位文件或文件夹的位置，并在资源管理器中将其打开。

跟我做

1．下载 Easy Explorer

可以到 http://sourceforge.net/project/showfiles.php?group_id=54542 下载 Easy Explorer 插

件，直接下载文件 org.sf.easyexplore_1.0.4.jar 即可。

2．安装 Easy Explorer

将 org.sf.easyexplore_1.0.4.jar 复制到%Eclipse%\plugins 目录下，重新启动 Eclipse，即可完成 Easy Explorer 插件的安装。

3．使用 Easy Explorer

Easy Explorer 插件正确安装后，在包资源管理器视图、大纲视图、编辑器、问题视图和导航器视图的右键菜单中增加【Easy Explorer】命令。

（1）右击项目“JavaApplication”，弹出右键快捷菜单，选择【Easy Explorer】命令，如图 3-37 所示。

（2）选择【Easy Explorer】命令后资源管理器会自动打开并定位到项目“JavaApplication”所在目录，如图 3-38 所示。在大纲视图、编辑器等视图中，尝试同样操作，体会 Easy Explorer 带来的方便。

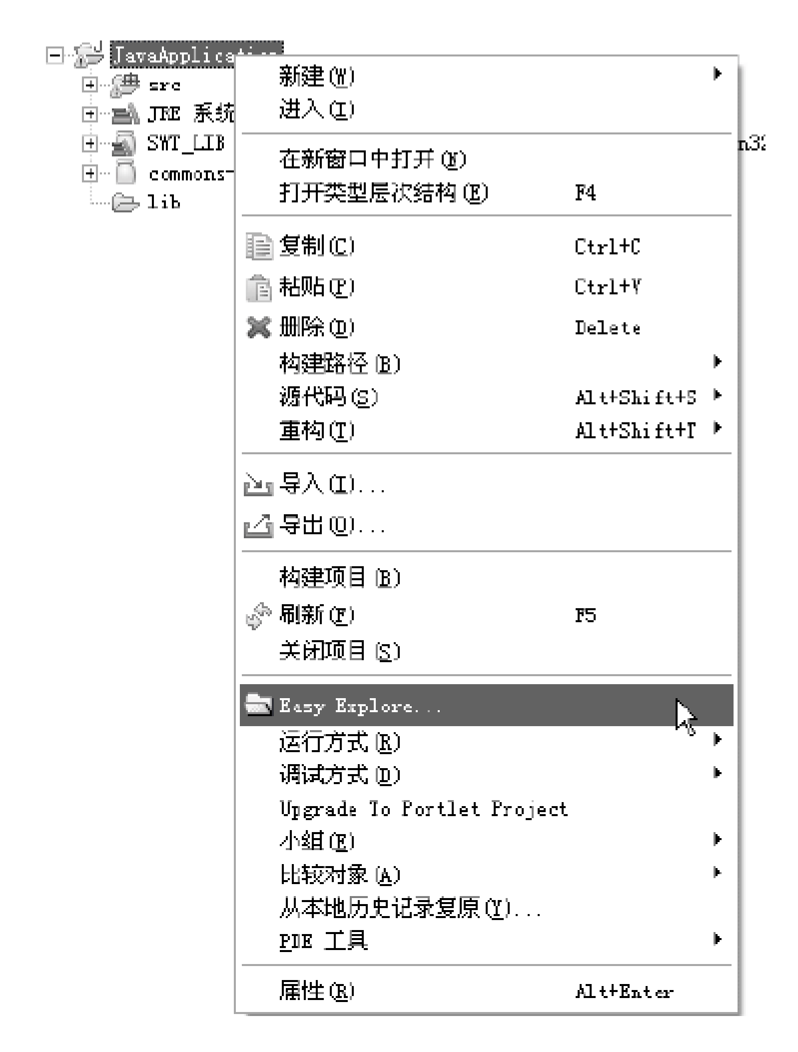


图 3-37　使用 Easy Explorer 打开项目



图 3-38　定位项目“JavaApplication”所在目录

（3）配置 Easy Explorer。单击【窗口】菜单，选择【首选项】命令，弹出【首选项】对话框。单击左侧列表的“Easy Explorer”选项，在右侧面板出现“Easy Explorer”配置信息，如图 3-39 所示。

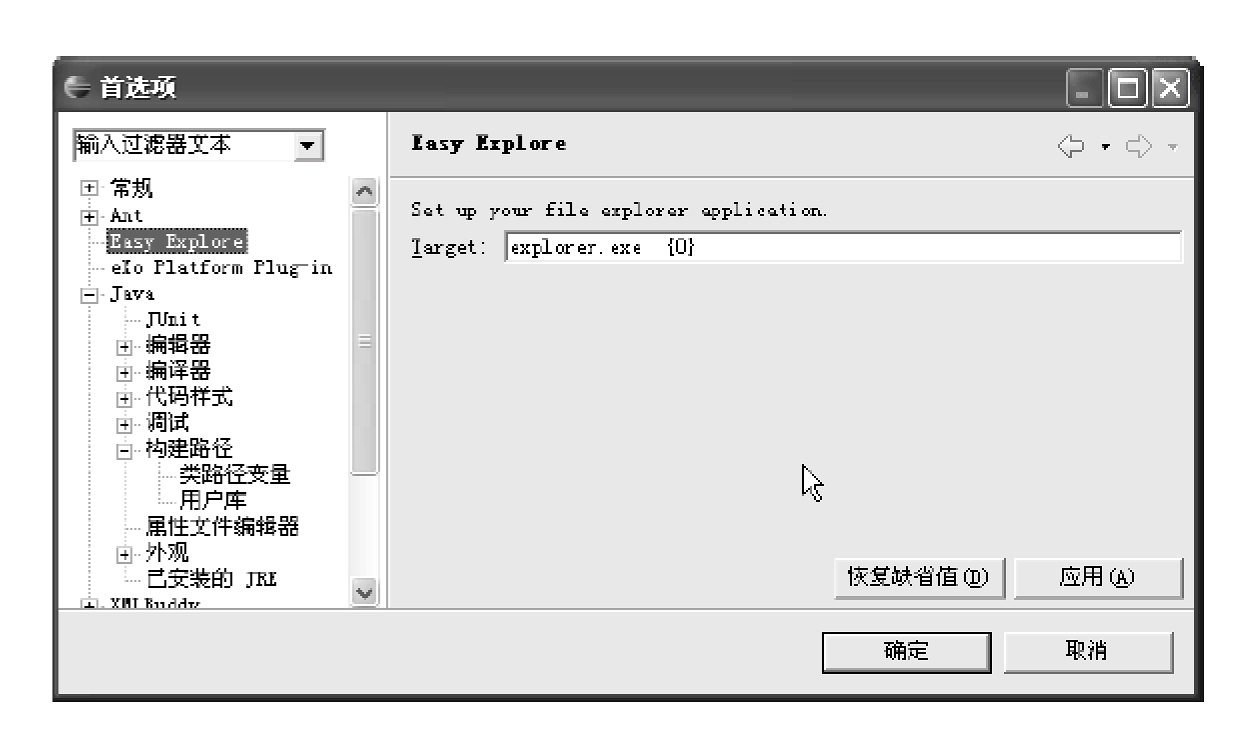


图 3-39　配置 Easy Explorer

可以看到，在 Windows 平台上是用 explorer.exe {0}来打开资源管理器的。explorer.exe 是 Windows 平台的系统命令，可以在命令行直接运行。{0}是传递给 explorer.exe 命令的参数，即要打开资源的路径信息。explorer.exe 命令格式是：Explorer [/n][/e][[,/root],[path]][[,/select],[path filename]]。

/n 表示以“我的电脑”方式打开一个新的窗口。

/e 表示以“资源管理器”方式打开一个新的窗口。

/root,[path]表示打开指定的文件夹，/root 表示只显示指定文件夹下面的文件（夹），不显示其他磁盘分区和文件夹，[path]表示指定的路径。

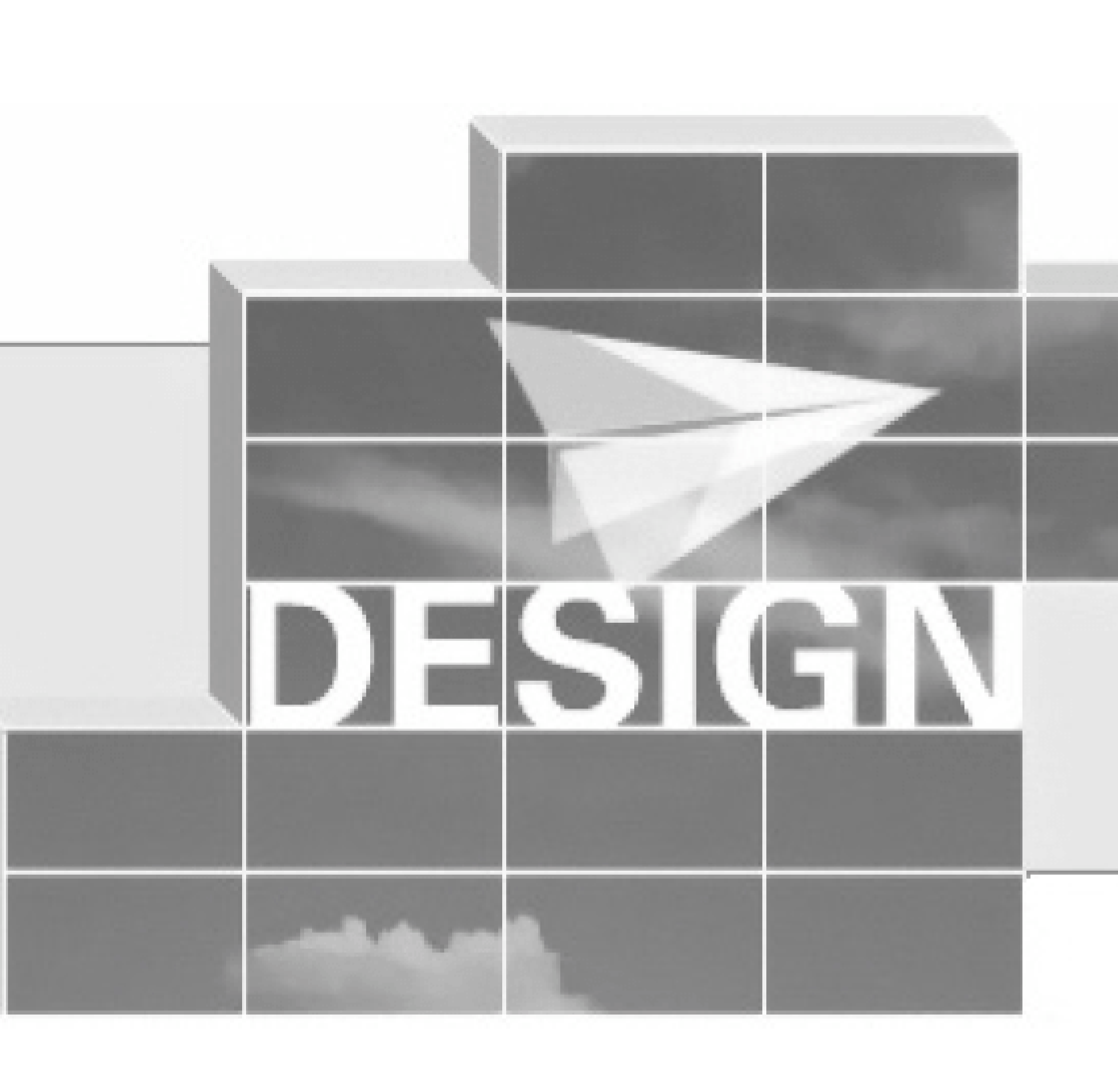
如果不加/root 参数，而只用[path]参数，则可以显示其他磁盘分区和文件夹中的内容。另外，[path]还可以指定网络共享文件夹。

/select,[path filename]表示打开指定的文件夹并且选中指定的文件，[path filename]表示指定的路径和文件名。

如果不加/select 参数，则系统会用相应的关联程序打开该文件。如果[path filename]不跟文件名就会打开该文件夹的上级目录并选中该文件夹。

以上参数都是可选的，读者可自由组合，体会 Easy Explorer 带来的方便。

第 2 篇



Web 开发技术实例详解

第 4 章　在 Eclipse 中进行资源构建——Ant 使用实例

Ant 是一种基于 Java 的 Build 工具，能非常方便地自动完成编译、测试、打包、部署等一系列任务，大大提高开发效率。理论上来说，它有些类似于（Unix）C 中的 Make，但没有 Make 的缺陷。Ant 也是一个跨平台的工具。本章首先介绍了 Ant 的基础知识，然后介绍了 Eclipse 与 Ant 的完美集成，接下来通过实例来说明在 Eclipse 中 Ant 的使用方法。

4.1　Ant 简介

Ant 是一个基于 java 的 Build 工具。大家都知道，现在已经有了许多的 Build 工具，例如 make、gnumake、nmake、jam 等，而且这些工具都非常优秀。那么为什么还要给大家介绍 Ant 这个新工具呢? 因为 Ant 是一个跨平台的 Build 工具。之所以 Ant 能跨平台，是因为 Ant 不再需要编写 shell 命令，Ant 的配置文件是基于 XML 的任务树，能让用户运行各种各样的任务，任务的运行是由实现了特定任务接口的对象来完成的。Ant 以 XML 的形式来编写构建脚本程序。运行 Ant 最重要的工作就是 Ant 的 build.xml 的编写。

4.1.1　构造文件的主要标记

1．project：在构建文件中声明一个项目

project 标记有 4 个属性，如表 4-1 所示：

表 4-1　project 标记的属性

属　性	描　述	是否必需
name	project 标记的名字	是
default	当没有显示指明目标时默认执行的目标	是
basedir	project 的基准目录。构造文件中所有的路径都以此为基准计算出来	是
description	project 的描述信息	否

每个 project 标记定义一个或多个 target，每个 target 又是想要执行的多个 task 的集合。在运行 Ant 时，可以将要执行的 target 以参数的形式给出，如果省略，则执行 project 标记的“default”属性所指定的 target。

2．target：在构建文件中声明一个目标

target 标记有如表 4-2 所示的属性：

表 4-2　target标记的属性

属　性	描　述	是否必需
name	目标的名字	是
depends	依赖的目标	否
if	仅当属性设置时才执行	否
unless	仅当属性没有设置时才执行	否
description	目标的描述信息	否

一个 target 可能依赖于别的 target。假设有 4 个 target，分别为 A、B、C、D，其中 D 的执行依赖于 B 和 C，C 依赖于 A，B 也依赖于 A，则其构建脚本如下所示：

```
<target name="A"/>
<target name="B" depends="A"/>
<target name="C" depends="A"/>
<target name="D" depends="B,C"/>
```

它们的依赖关系如图 4-1 所示，Ant 会自动处理这种依赖关系。

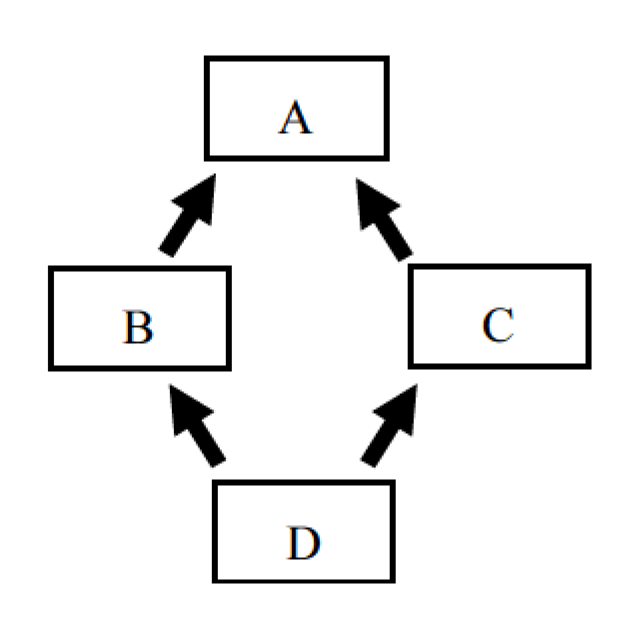


图 4-1　4 个目标之间的依赖关系

3．task：在构建文件中声明一个任务

task 标记定义了能够执行的程序代码。一个 task 可以有多个属性，其通常的结构如下：

```
<name attribute1="value1" attribute2="value2" ... />
```

其中“name”是 task 的名字，“attributeN”是该属性的名字，“valueN”是该属性的值。

4．property：在构建文件中声明一个属性

一个 project 可以通过 property 标记设定多个属性，每个属性有“name”和“value”两部分组成。对属性的引用可以通过将属性的名字放在“${”和“}”之间的方式实现。例如，如果属性“builddir”的值为“build”，其他属性的值如果设置为“${builddir}/classes”，则最终将被解析为“build/classes”。

4.1.2　Ant 的常用任务（Task）

1．mkdir：创建目录

该标记用于创建一个目录。例如：

```
<mkdir dir="builddir"/>
```

创建名字为“builddir”的目录。

2．jar：将若干文件打包成 jar 文件

该标记用于将若干个文件打包成一个 jar 文件。其常用属性如表 4-3 所示：

表 4-3　jar标记的属性

属　性	描　述	是否必需
destfile	生成的 Jar 文件	是
basedir	被归档的文件目录	否
includes	被归档的文件模式。如果忽略则包括所有的文件	否
excludes	被排除的文件模式。如果忽略则不忽略任何文件	否

```
<jar destfile="${dist}/lib/app.jar" basedir="${build}/classes"/>
```

将“${build}/classes”目录下的所有文件归档到“${dist}/lib”目录下的 app.jar 文件中。

```
<jar destfile="${dist}/lib/app.jar"
        basedir="${build}/classes"
        excludes="**/Test.class"
/>
```

将位于“${build}/classes”目录下，除 Test.class 之外的所有文件打包成 app.jar 文件，并将其放到“${dist}/lib”目录下。

```
<jar destfile="${dist}/lib/app.jar"
        basedir="${build}/classes"
        includes="mypackage/test/**"
        excludes="**/Test.class"
/>
```

将位于“${build}/classes/mypackage/test”目录下，除 Test.class 之外的所有文件打包成 app.jar 文件，并将其放到“${dist}/lib”目录下。

3．javac：编译 java 源文件

该标记用于编译 Java 文件。其常用属性如表 4-4 所示：

表 4-4　javac标记的属性

属　　性	描　　述	是 否 必 需
srcdir	包含所有要编译的 Java 文件的目录	是
destdir	存放生成的 class 文件的目录	否
includes	被包含的 Java 文件的模式。如果省略则表示包含所有的 Java 文件	否
excludes	被排除的 Java 文件的模式	否
classpath	类路径	否
debug	是否包含调试信息	否

4. java：执行 java 程序

该标记用于执行 Java 程序。其常用属性如表 4-5 所示：

表 4-5　java标记的属性

属　　性	描　　述	是 否 必 需
classname	要运行的 Java Class 的名字	是
jar	包含要运行的 Java Class 的 Jar 文件名	是
classpath	所用到的 Classpath	否

5. javadoc：生成程序的 API 文档

该标记利用 javadoc 工具生成程序代码的文档。其常用属性如表 4-6 所示：

表 4-6　javadoc标记的属性

属　　性	描　　述	是 否 必 需
destdir	生成文件的存放目录	是
sourcepath	查找源文件的路径	是
classpath	查找用户类的路径	否

6. echo：回显信息

该标记将在 System.out 输出信息或者将信息写入文件。其常用的属性如表 4-7 所示：

表 4-7　echo标记的属性

属　　性	描　　述	是 否 必 需
message	要显式的信息	是
file	将显式信息写入的文件	否
append	是否以追加的方式写入文件	否
level	信息的报告级别。包括“error”、“warning”、“info”、“verbose”和“debug”	否

7．property：声明一个属性

该标记在 project 中定义一个属性。其常用的属性如表 4-8 所示：

表 4-8　property标记的属性

属　　性	描　　述	是 否 必 需
name	属性的名字	否
value	属性的值	是

```
<property name="foo.dist" value="dist"/>
```

定义了名字为“foo.dist”的属性，其值为“dist”。

```
<property file="foo.properties"/>
```

读取在“foo.properties”文件中定义的属性。

4.2　Eclipse 与 Ant 的集成

Eclipse 的外部工具框架集成了 Ant，可以在 Eclipse 中直接运行 Ant。

4.2.1　创建 Ant 构建文件

Ant 构建文件是简单的 XML 文件，本节将介绍如何在 Eclipse 中创建一个构建文件。

跟我做

（1）启动 Eclipse，新建一个名字为“AntExample”的 Java 工程。

（2）右击“AntExample”工程，在快捷菜单中选择【新建】|【文件】命令，打开【新建文件】窗口。

（3）在【文件名】文本框中输入“build.xml”，单击【完成】按钮，在 AntExample 工程的根目录下创建 build.xml 文件。

注意：输入文件可以为任何名称，只要确保它具有.xml 扩展名即可，这里选用“build.xml”，只是为了增加可交流性。

4.2.2　编辑 Ant 构建文件

因为 Ant 构建文件是 XML 文件，所以可以用任何文本编辑器来编辑它们。但是，Eclipse 提供的 Ant 编辑器具有语法着色、内容辅助、出现标记和大纲视图等优点。本节介绍如何通过 Eclipse Ant 编辑器来编辑 Ant 构建文件。

跟我做

（1）右击 AntExample 工程的 build.xml 文件，在快捷菜单中选择【打开方式】|【Ant 编辑器】命令，将 build.xml 用 Eclipse Ant 编辑器打开。

注意： 在.xml 文件包含构建文件内容之前，它的默认编辑器是一个简单文本编辑器，但是可以在窗口>首选项>常规>文件关联中更改它。

（2）在该编辑器中输入如下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!--创建一个名字为 AntExample 的项目，缺省目标是 firstTarget，参加路径是当前目录-->
<project name="AntExample" default="secondTarget" basedir=".">
    <!--声明一个属性，名字为 firstText，值为 first-->
    <property name="firstText" value="first" />
    <!--声明一个属性，名字为 secondText，值为 second-->
    <property name="secondText" value="second" />
    <!--创建一个名字为 firstTarget 的目标-->
    <target name="firstTarget">
        <!--创建一个 echo 任务，将属性 firstText 的值输出到 System.out 上去-->
        <echo>${firstText}</echo>
    </target>
    <!--创建一个名字为 secondTarget 的目标-->
    <target name="secondTarget">
        <!--创建一个 echo 任务，将属性 firstText 的值输出到 System.out 上去-->
        <echo>${WorldText}</echo>
    </target>
</project>
```

Eclipse Ant 编辑器具有强大的内容辅助功能，例如当输入“<t”开始的第二个目标时，同时按下“Alt”和“/”键激活内容辅助，如图 4-2 所示，将显示一列有效的补全。利用键盘的上下箭头选择<target>补齐代码，编辑器同时插入开始标记和结束标记，然后将光标定位在适当的位置以便输入此标记的属性。

另外，大纲视图以树形结构组织 build.xml 文件中的每个属性和每个目标，如图 4-3 所示，具有很强的可读性。



图 4-2　Ant 编辑器的代码辅助功能



图 4-3　大纲视图

（3）保存“build.xml”文件，完成对该文件的编辑。

4.2.3　运行 Ant 构建文件

带有 xml 扩展名的任何文件都可以作为 Ant 构建文件运行。当然，并非所有这种文件都是真正的 Ant 构建文件，但是，如果错误地尝试将非 Ant.xml 文件作为 Ant 构建文件运行，也不会产生什么危害。本小节介绍在 Eclipse 中运行 Ant 构建文件。

跟我做

（1）右击“AntExample”工程的“build.xml”，在快捷菜单中选择【运行方式】|【Ant 构建…】命令，打开启动配置对话框，如图 4-4 所示。

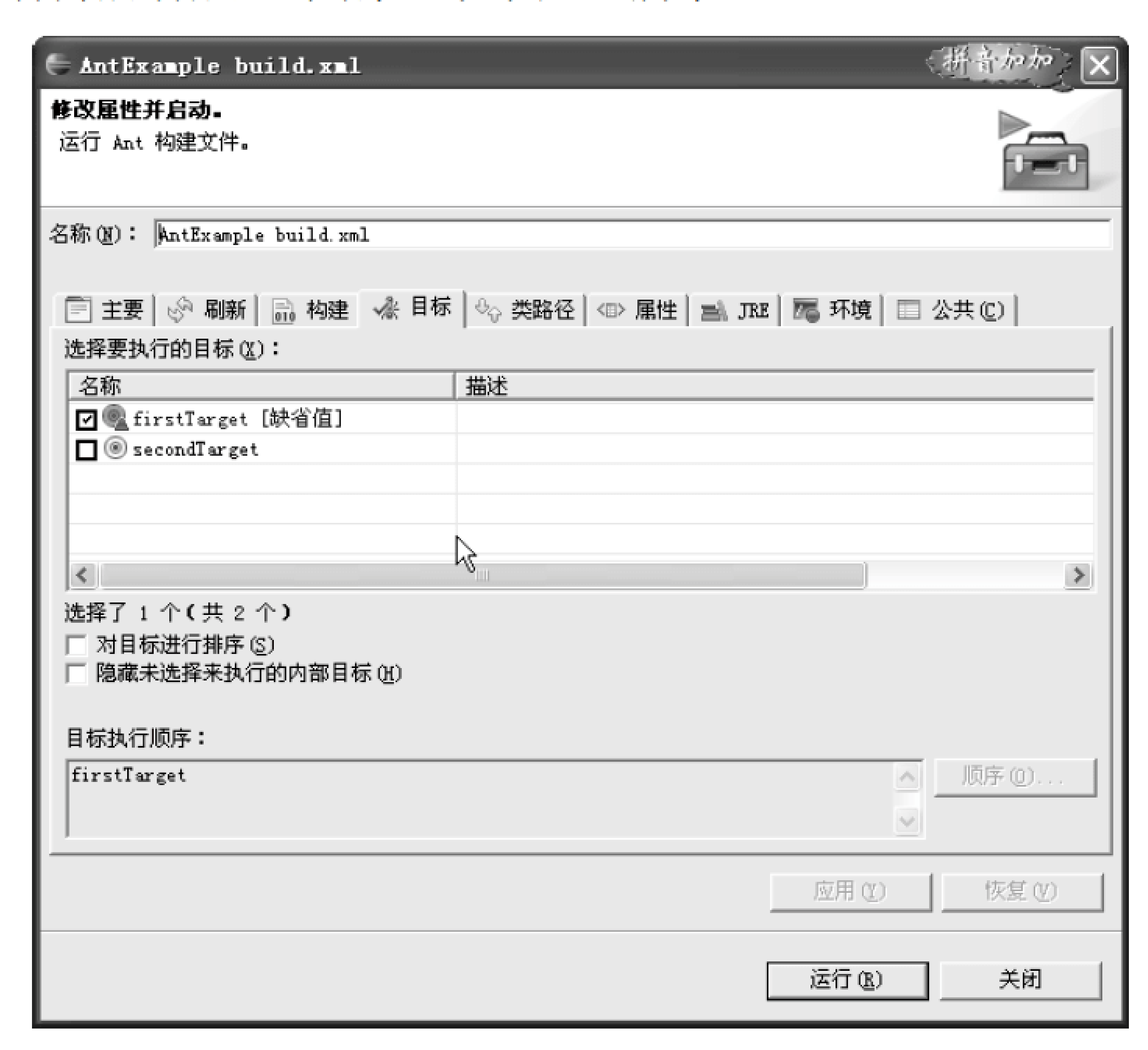


图 4-4　启动配置对话框

注意： 在启动配置对话框中可以配置 Ant 构建文件运行方式的许多方面，这里只选择要运行的 Ant 目标及其顺序。

（2）单击【运行】按钮，运行 Ant 构建文件，控制台视图出现如图 4-5 所示信息。

```
Buildfile: E:\Eclipsebook\eclipse\workspace\AntExample\build.xml
firstTarget:
     [echo] first
BUILD SUCCESSFUL
Total time: 160 milliseconds
```

图 4-5　运行 firstTarget 控制台输出信息

4.2.4　使用 Ant 视图

Eclipse 提供的 Ant 视图可以在一个位置使用所有的 Ant 构建文件，本小节介绍如何使

用 Ant 视图。

跟我做

（1）在 Eclipse 菜单中选择【窗口】|【显示视图】|【Ant】命令，打开 Ant 视图。

（2）单击【添加构建文件】按钮，打开如图 4-6 所示的【选择构建文件】窗口。

（3）选择“build.xml”文件，单击【确定】按钮，将其加入到 Ant 视图中，如图 4-7 所示。



图 4-6　【选择构建文件】窗口



图 4-7　Ant 视图

（4）选中“secondTarget”目标，并单击【运行所选择的目标】按钮，执行“secondTarget”目标。输出信息如图 4-8 所示。

```
Buildfile: E:\Eclipsebook\eclipse\workspace\AntExample\build.xml
secondTarget:
      [echo] second
BUILD SUCCESSFUL
Total time: 160 milliseconds
```

图 4-8　运行 secondTarget 控制台输出信息

（5）在 Ant 视图中右击 AntExample，在快捷菜单中选择【打开方式】|【Ant 编辑器】命令，打开“build.xml”构建文件。

（6）在 Ant 视图中右击 AntExample，在快捷菜单中选择【运行方式】|【Ant 构建…】命令，打开【启动配置】对话框。可以修改构建文件在 Ant 视图中运行的方式。

（7）在 Ant 视图中选中 AntExample，然后单击【除去所选择的构建文件】按钮，可以将 AntExample 构建文件从 Ant 视图中除去。

4.3　用 build.xml 编写 Ant 部署文件实例

在了解了 Ant 的基本知识以及 Eclipse 与 Ant 的集成之后，本节通过一个具体的实例来说明如何编写 Ant 构建文件。

4.3.1　编写 build.xml 文件之前的准备

在前面建立的 AntExample 工程基础上，创建几个目录，为编写 build.xml 文件做准备。

跟我做

（1）右击“AntExample”工程，在快捷菜单中选择【新建】|【源文件夹】命令，打开【新建源文件夹】窗口。

（2）在【文件夹名】文本框中输入“src”，单击【确定】按钮，创建“src”源文件夹，用来存放 Java 源文件。

（3）右击“AntExample”工程，在快捷菜单中选择【新建】|【文件夹】命令，打开【新建文件夹】窗口。

（4）在【文件夹名】文本框中输入“classes”，单击【确定】按钮，创建“classes”文件夹，用来存放编译后的 class 文件。

（5）用同样的方法创建 dist 文件夹存放打包后的 jar 文件，lib 文件夹存放编译和运行用到的所有 jar 文件，doc 存放 API 文档。

（6）右击“AntExample”工程，在快捷菜单中选择【新建】|【文件】命令，打开【新建文件】窗口。

（7）在【文件名】文本框中输入“build.xml”，单击【确定】按钮，创建 build.xml 文件。

（8）再次打开【新建文件】窗口。

（9）在【文件名】文本框中输入“example.properties”，单击【确定】按钮，创建 example.properties 文件，最后形成如图 4-9 所示的工程结构。

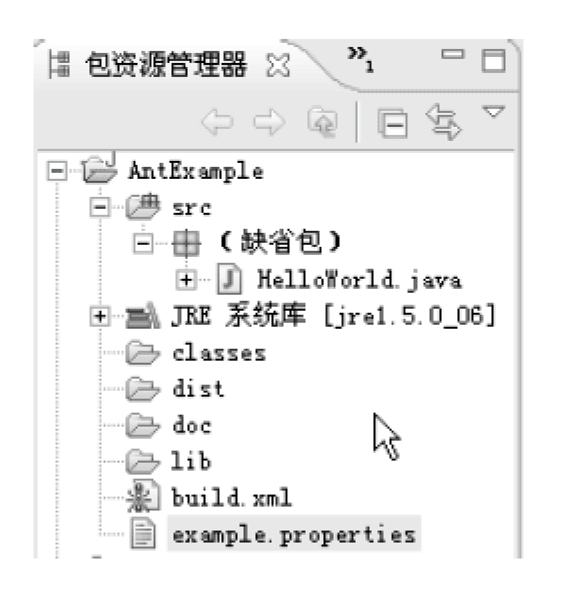


图 4-9　AntExample 工程的目录结构

4.3.2　使用 property 定义属性实例

通过 property 任务为 project 定义属性，本小节介绍常用的两种定义属性的方式：直接定义法和读取属性文件法。

跟我做

（1）打开“example.properties”文件输入如下代码：

```
classes.dir=classes
lib.dir=lib
dist.dir=dist
doc.dir=doc
```

（2）编辑 build.xml 文件，创建一个名为“AntExample”的 project，输入如下代码：

```
<!--创建一个名字为 AntExample 的 project-->
<project name="AntExample">
</project>
```

（3）创建名字为“src.dir”，值为“src”的属性。在 build.xml 文件中输入如下代码：

```
<property name="src.dir" value="src" />
```

该属性定义方法为直接定义法，是最简单的一种方法，不过当定义的属性过多时，构建程序的可读性会下降。

（4）将 example.properties 文件中定义的属性读入，在 build.xml 文件中输入如下代码：

```
<property file="example.properties" />
```

该属性定义方法为属性文件法。

4.3.3　生成 Java 实例程序

为了便于后续章节任务的说明，这里建立经典的“Hello World”Java 程序。

跟我做

（1）在“AntExample”工程中建立“HelloWorld.java”文件。
（2）编辑 HelloWorld.java 文件，在该文件中输入如下代码：

```
public class HelloWorld {
    //HellWorld 类的入口方法
    public static void main(String[] args) {
        //往控制台视图输出 HelloWorld 字符串
        System.out.println("Hello world!");
    }
}
```

4.3.4　使用编译任务编译 Java 类实例

javac 任务用于编译 Java 类。本小节讲解如何利用 javac 任务将 HelloWorld.java 编译，并将生成的 HelloWorld.class 文件放入到 classes 文件夹中。

跟我做

（1）在“build.xml”中创建名字，为“compile”的 target，输入如下代码：

```
<!--创建名字为 compile 的 target-->
<target name="compile" description="compile the HelloWorld.java file">
        <!--将 src 目录下的所有 java 文件编译，并将生成的 class 文件放到 classes 目录下-->
        <javac srcdir="${src.dir}" destdir="${classes.dir}">
         </javac>
</target>
```

（2）右击“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台输出如图 4-10 所示信息。

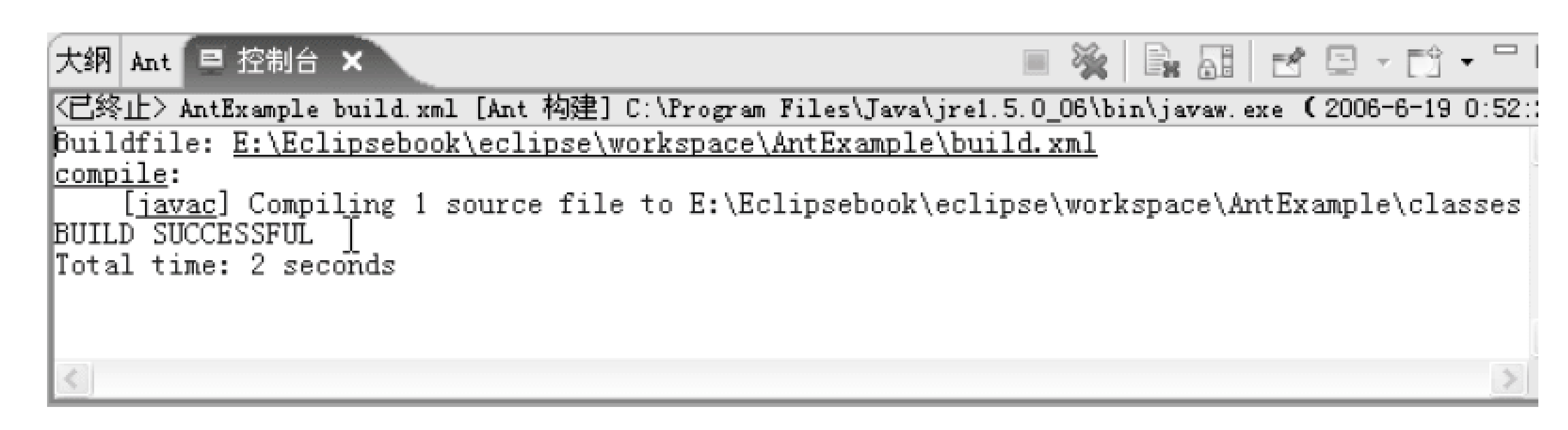


图 4-10　运行 compile 目标后的输出信息

（3）右击“AntExample”工程的“classes”文件夹，在快捷菜单中选择【刷新】命令，即可看到 classes 目录下已经存在 HelloWorld.class 文件了。

4.3.5　使用 Java 任务执行 Java 类实例

Java 任务可以执行指定的 Java 类。本小节讲解如何利用 java 任务执行 4.3.4 小节编译成的 HelloWorld.class。

跟我做

（1）在 build.xml 中创建名字为 run 的 target，输入如下代码：

```
<!--创建名字为 run 的 target，其依赖于 compile 目标-->
<target name="run" description="run the HelloWorld.class" depends="compile">
          <!--执行 HelloWorld 类，classname 属性指定类的名字-->
          <java classname="HelloWorld">
               <!--指定 Classpath-->
               <classpath>
                    <!--将 classes 目录设定为 classpath-->
                    <pathelement path="${classes.dir}" />
               </classpath>
          </java>
</target>
```

（2）右击“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台输出如图 4-11 所示信息。

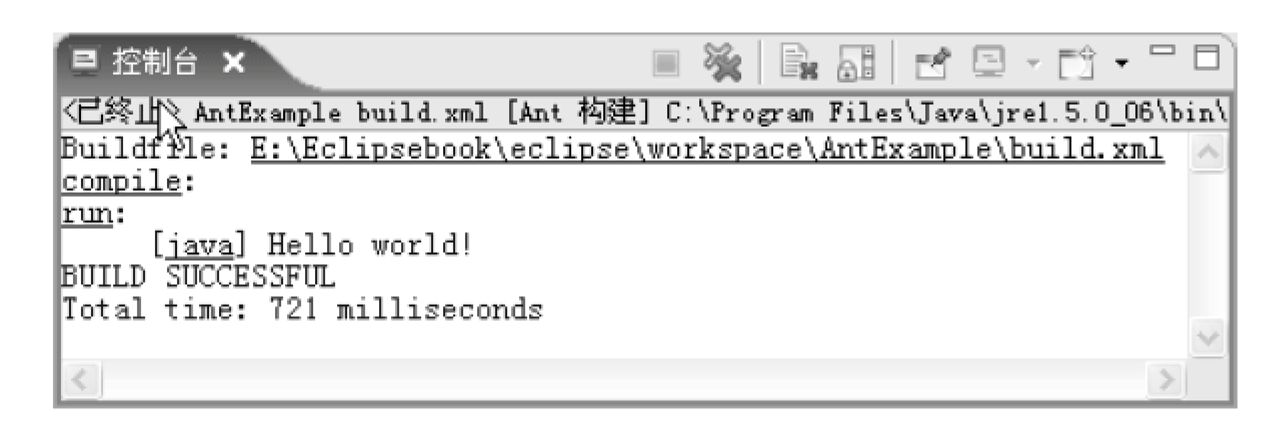


图 4-11　运行 run 目标后的控制台输出信息

4.3.6　使用 jar 任务打包文件实例

jar 任务可以将若干个文件打包成 jar 文件。本小节讲解如何利用 jar 任务将 classes 目录下的所有 class 打包成 jar 文件。

跟我做

（1）在“build.xml”文件中创建名字为 pack 的 target，输入如下代码：

```
<!--创建名字为 pcak 的 target，其依赖于 run target-->
<target name="pack" depends="run" description="make .jar file">
        <!--将 classes 目录下的所有文件打包为 hello.jar，并存放到 dist 目录下-->
        <jar destfile="${dist.dir}/hello.jar" basedir="${classes.dir}">
        </jar>
</target>
```

（2）右击“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台输出如图 4-12 所示信息。

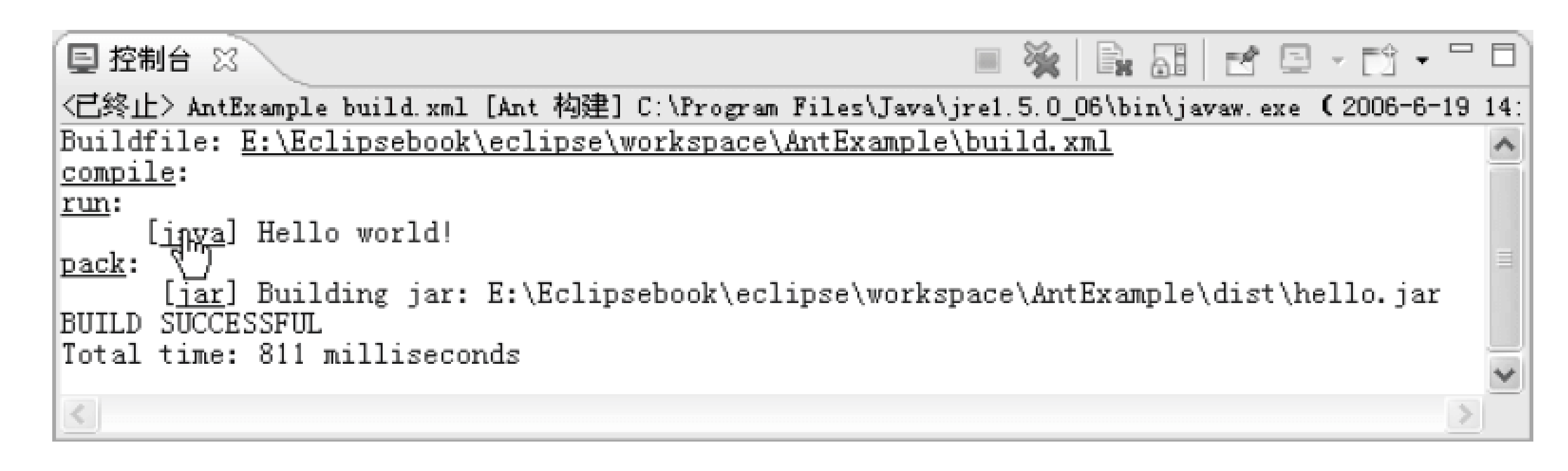


图 4-12　运行 pack 目标后控制台的输出信息

4.3.7　使用 javadoc 任务生成文档实例

javadoc 任务可以生成 Java 类的 API 文档。本小节讲解利用 javadoc 生成 src 文件夹下所有 Java 类的 API 文档。

跟我做

（1）在“build.xml”中创建名字为 doc 的 target，输入如下代码：

```
<!--创建名字为 doc 的 target，生成 src 目录下的所有类的 API 文档-->
<target name="doc" depends="pack" description="create api doc">
        <!--javadoc 任务，将生成的 API 文档放到 doc 目录下，window 的 title 设定为
```

```
HelloWorldAPI-->
            <javadoc destdir="${doc.dir}" author="true" version="true" use="true" windowtitle
="HelloWorld API">
                <!--设定 src 目录下的所有 Java 程序将被生成 API 文档-->
                <fileset dir="${src.dir}" defaultexcludes="yes">
                </fileset>
                <doctitle>
                      <![CDATA[<h1>Hello, test</h1>]]>
                </doctitle>
                 <bottom>
                <![CDATA[<i>All Rights Reserved.</i>]]>
                </bottom>
           </javadoc>
</target>
```

（2）右击“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台输出如图 4-13 所示信息。

（3）右击“AntExample”工程的“doc”文件夹，在快捷菜单中选择【刷新】命令，可以看到上一步生成的 API 文档，如图 4-14 所示。

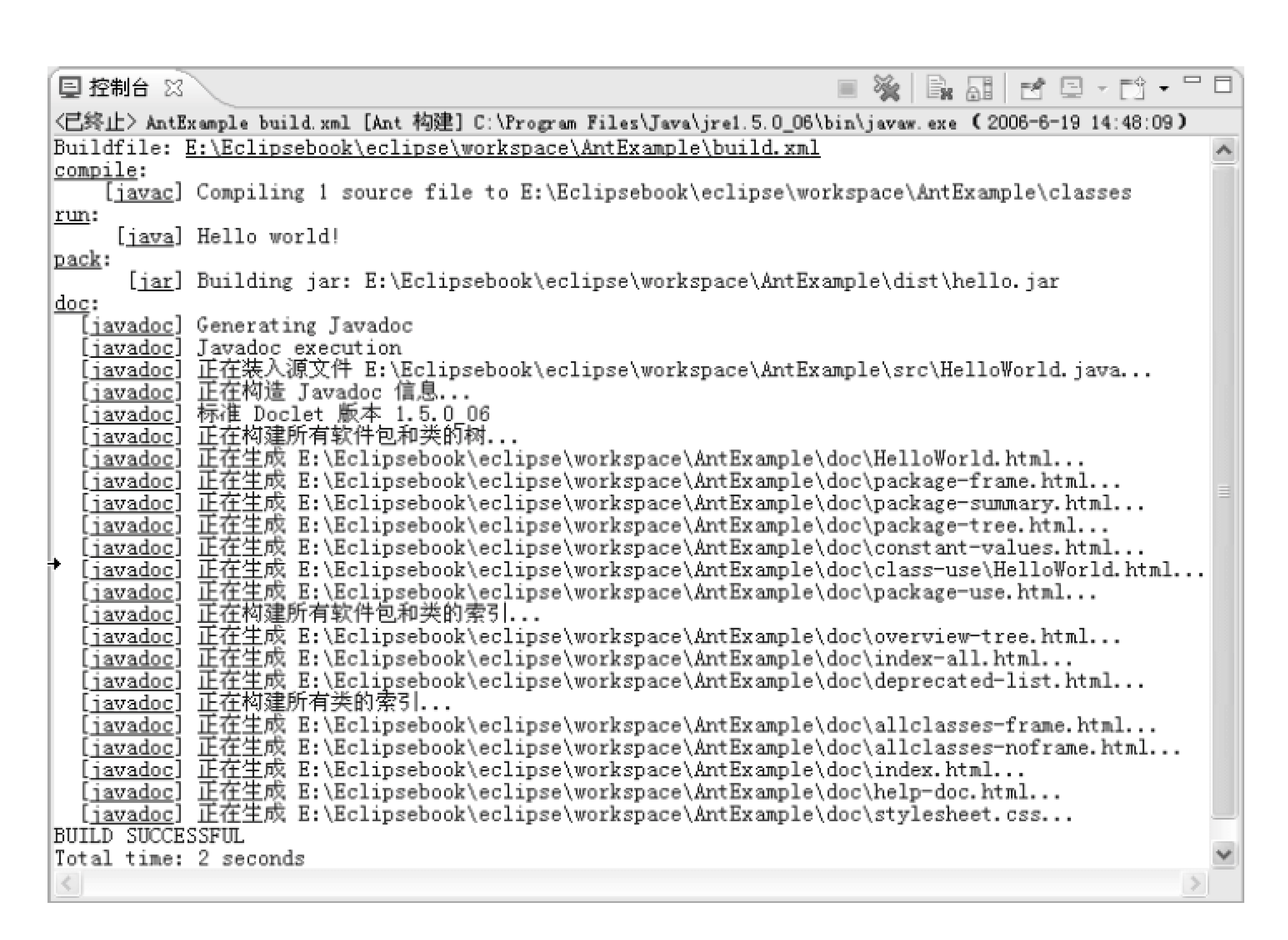


图 4-13　执行 doc 目标后的控制台输出信息

包资源管理器　层次结构
AntExample
src
(缺省包)
HelloWorld.java
JRE 系统库 [jre1.5.0_06]
classes
HelloWorld.class
dist
hello.jar
doc
class-use
HelloWorld.html
resources
allclasses-frame.html
allclasses-noframe.html
constant-values.html
deprecated-list.html
HelloWorld.html
help-doc.html
index.html
index-all.html
overview-tree.html
package-frame.html
package-list
package-summary.html
package-tree.html
package-use.html
stylesheet.css
lib
build.xml
example.properties
ScoreWeb

图 4-14　执行 doc 后的工程目录结构

4.3.8　使用 mail 任务发送电子邮件实例

mail 任务可以用来发送 SMTP 电子邮件。本小节讲解通过 mail 任务发送电子邮件，并且将 src 文件夹下的所有 java 文件作为邮件的附件。

在讲解之前需要准备发送方邮件地址，假设为 sender@tom.com，用户名为“demo”，

密码为“demo”；发送方邮件服务器的地址为“smtp.tom.com”；接收方的邮件地址，假设为 receiver@tom.com。

跟我做

（1）在 build.xml 中创建名字为 mail 的 target，输入如下代码：

```
<!--创建名字为 mail 的 target-->
<target name="mail" description="mail somthing">
<!--mialhost 设定为 smtp.tom.com，用户名和密码都是 demo，主题是 Test build ，字符集为 utf-8-->
    <mail    mailhost="smtp.tom.com"    user="demo"    password="demo"    subject="Test    build"
charset="utf-8">
        <!--发送方邮件地址-->
        <from address="sender@tom.com" />
        <!--接收方邮件地址-->
        <to address="receiver@tom.com" />
        <!--邮件内容-->
        <message>The    nightly build has completed</message>
        <!--邮件附件包含 src 文件中的所有 java 文件-->
        <fileset dir="${src.dir}">
              <include name="**/*.java" />
        </fileset>
    </mail>
</target>
```

（2）到 http://java.sun.com/products/javabeans/jaf/downloads/index.html 下载 JAF1.1，解压后其目录结构如图 4-15 所示，将其中的 activation.jar 复制到 AntExample 工程的 lib 目录下。

（3）到 http://java.sun.com/products/javamail/downloads/index.html 上下载 JavaMail1.4，解压后其目录结构如图 4-16 所示，将其中的 mail.jar 和 lib 目录下的所有 jar 文件复制到 AntExample 工程的 lib 目录下。

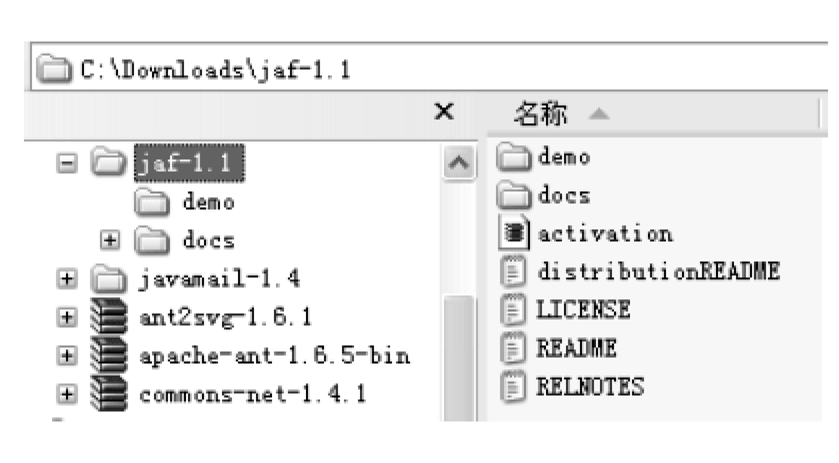


图 4-15　JAF 解压后的目录结构

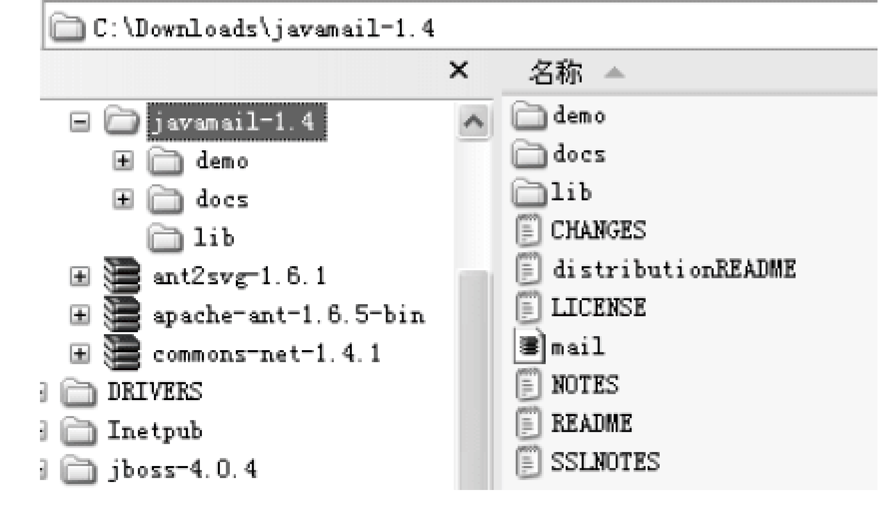


图 4-16　JavaMail 解压后的目录结构

（4）右击“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建…】命令，打开【AntExample build.xml】窗口。

（5）选择【目标】选项卡，在【选择要执行的目标】列表中选择【mail】选项。

（6）选择【类路径】选项卡，配置类路径，然后单击【添加 JAR】按钮，打开【选择 JAR】窗口。

（7）展开 AntExample 工程树结构，选中工程 lib 目录下的所有 jar 文件，单击【确定】按钮，将上述选择的 jar 文件加入到类路径上。

（8）单击窗口下方的【运行】按钮，控制台输出如图 4-17 信息。

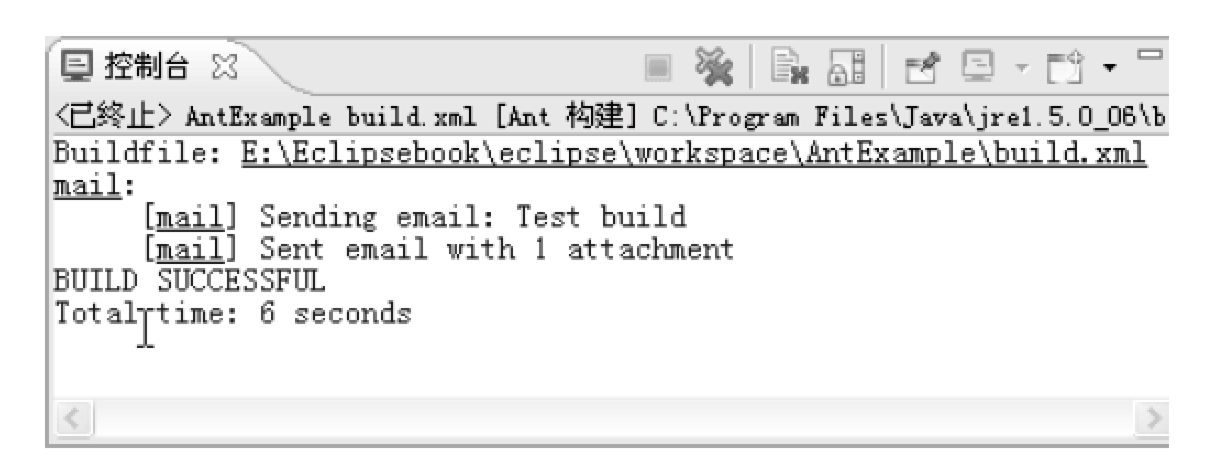


图 4-17　运行 mail 任务后控制台的输出信息

4.4　生成构建器

本节介绍如何创建项目构建器，以便运行 Ant 构建文件。

跟我做

（1）右击“AntExample”工程，在快捷菜单中选择【属性】命令，打开【AntExample 的属性】窗口。

（2）在窗口左侧的树形结构中选择【构建器】选项，窗口右侧出现构建器配置环境，单击【新建】按钮，打开如图 4-18 所示的【选择配置类型】窗口。



图 4-18　【选择配置类型】窗口

（3）选择【Ant 构建】选项，单击【确定】按钮，打开【新构建器的属性】窗口。

（4）在【名称】文本框中输入“AntExample”，在【主要】选项卡中单击【构建文件】组中的【浏览工作空间】按钮，打开【选择位置】对话框，如图 4-19 所示。

（5）选择“build.xml”文件，单击【确定】按钮，将选择构建文件为“${workspace_loc:/AntExample/build.xml}”。

（6）单击【基本目录】组中的【浏览工作空间】按钮，打开【选择文件夹】对话框，如图 4-20 所示。

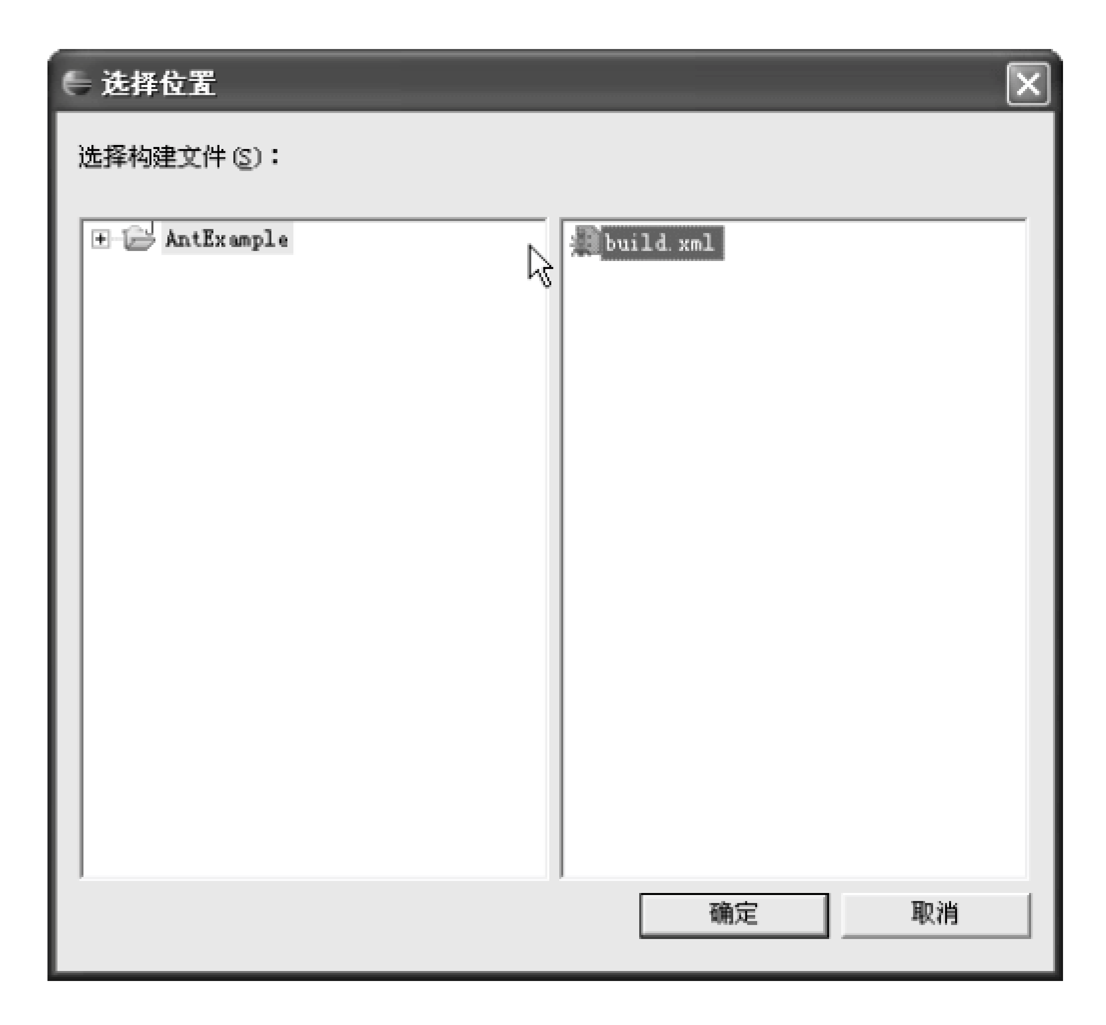



图 4-19　【选择位置】对话框



图 4-20　【选择文件夹】对话框

（7）选择“AntExample”文件，单击【确定】按钮，将【基本目录】设置为“${workspace_loc:/AntExample}”，如图 4-21 所示。

（8）在【刷新】选项卡中，选择【完成时刷新资源】复选框，在作用域变量的列表中选择【包含所选资源的项目】单选按钮，如图 4-22 所示。



图 4-21　【主要】选项卡的配置信息

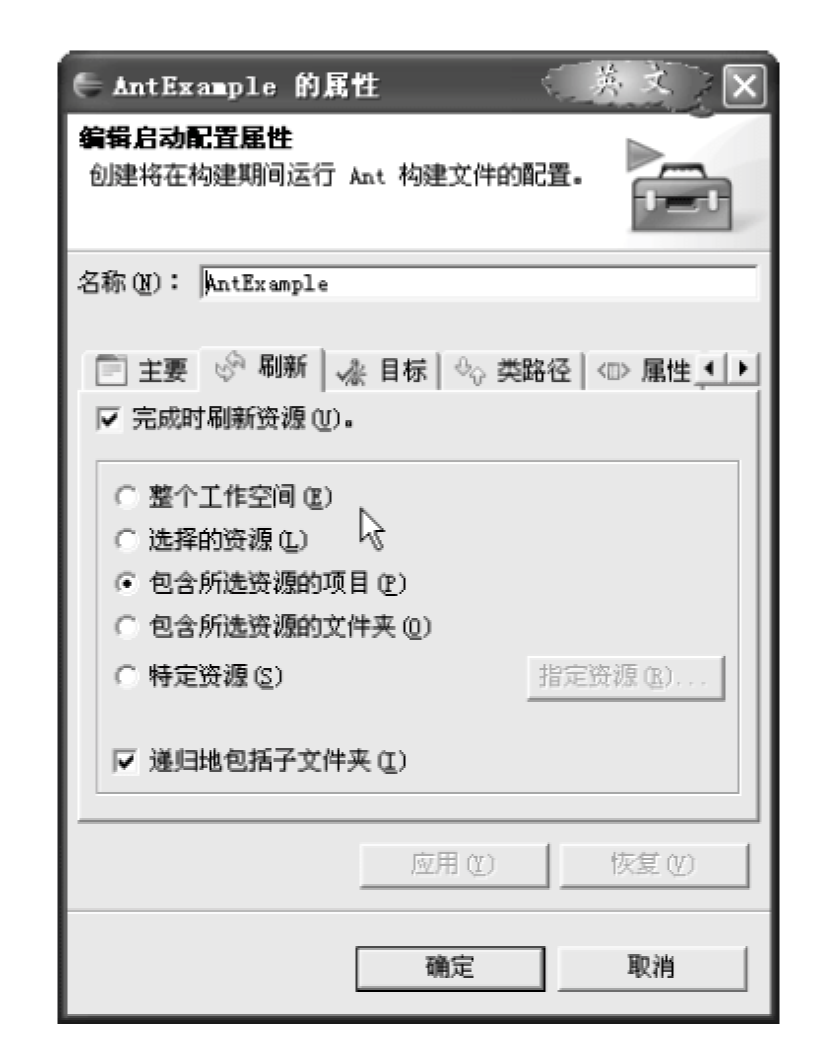


图 4-22　【刷新】选项卡的配置信息

注意： 默认情况下，当项目构建器完成运行时不执行任何刷新，所以进行了如上设置。

（9）在【目标】选项卡中，单击【自动构建】右边的【设置目标】按钮，打开如图 4-23 所示的【设置目标】对话框。

（10）在【选择要执行的目标】列表中选择“doc”目标，单击【确定】按钮，设定【自动构建】的默认目标为“doc”，如图 4-24 所示。



图 4-23　【设置目标】对话框

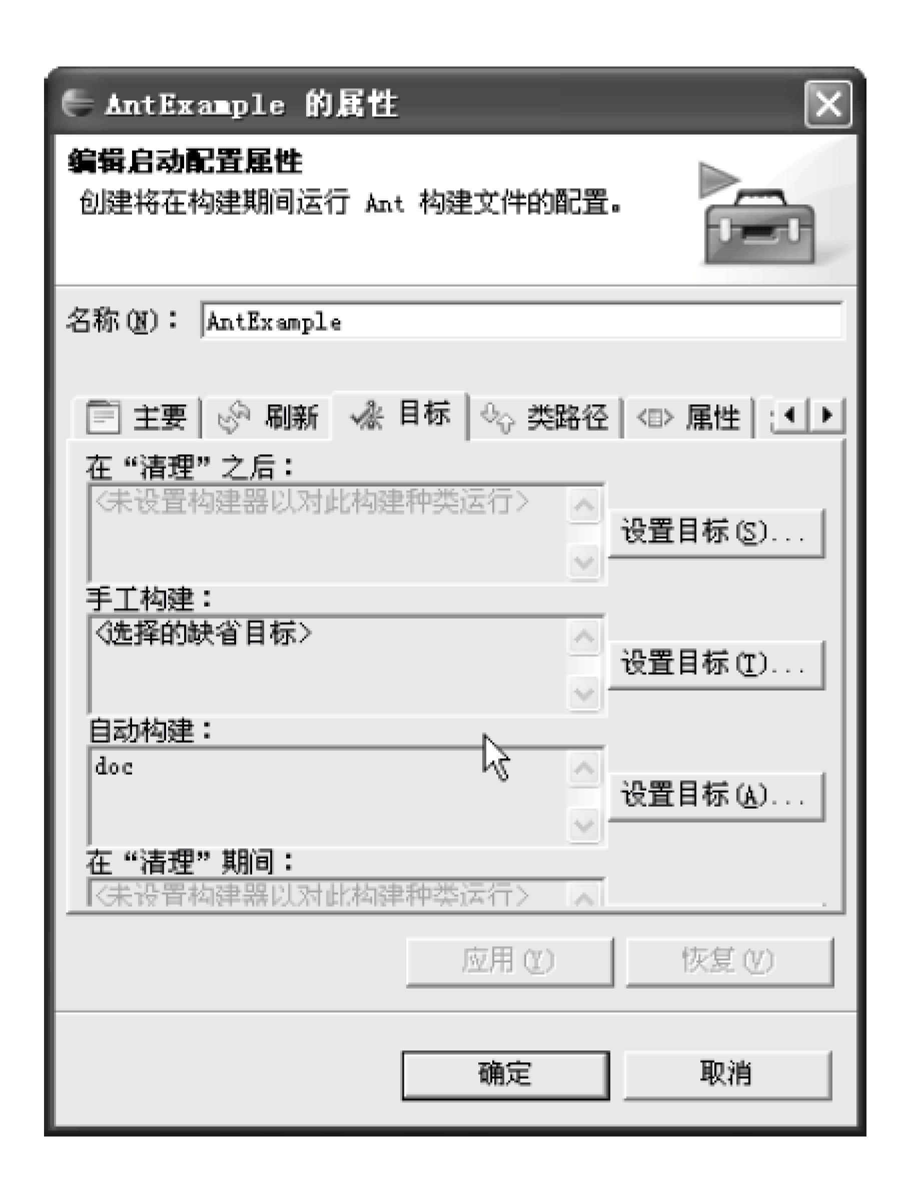


图 4-24　设置【自动构建】的默认目标为"doc"

（11）单击【确定】按钮，保存项目构建器，在如图 4-25 所示的【为项目配置构建器】中增加了 AntExample。



图 4-25　增加了 AntExample 构建器

4.5　执行构建

项目构建器的核心问题是它们不由用户显式运行，而是在对拥有构建文件的项目限定的构建发生的任何时间运行。

跟我做

（1）打开 HelloWorld.java 做一些细小的更改并保存该更改。

（2）由于在 AntExample 项目构建器中设定自动构建将执行 doc 目标，所以此时 doc 目标被执行，控制台视图输出信息如图 4-26 所示。

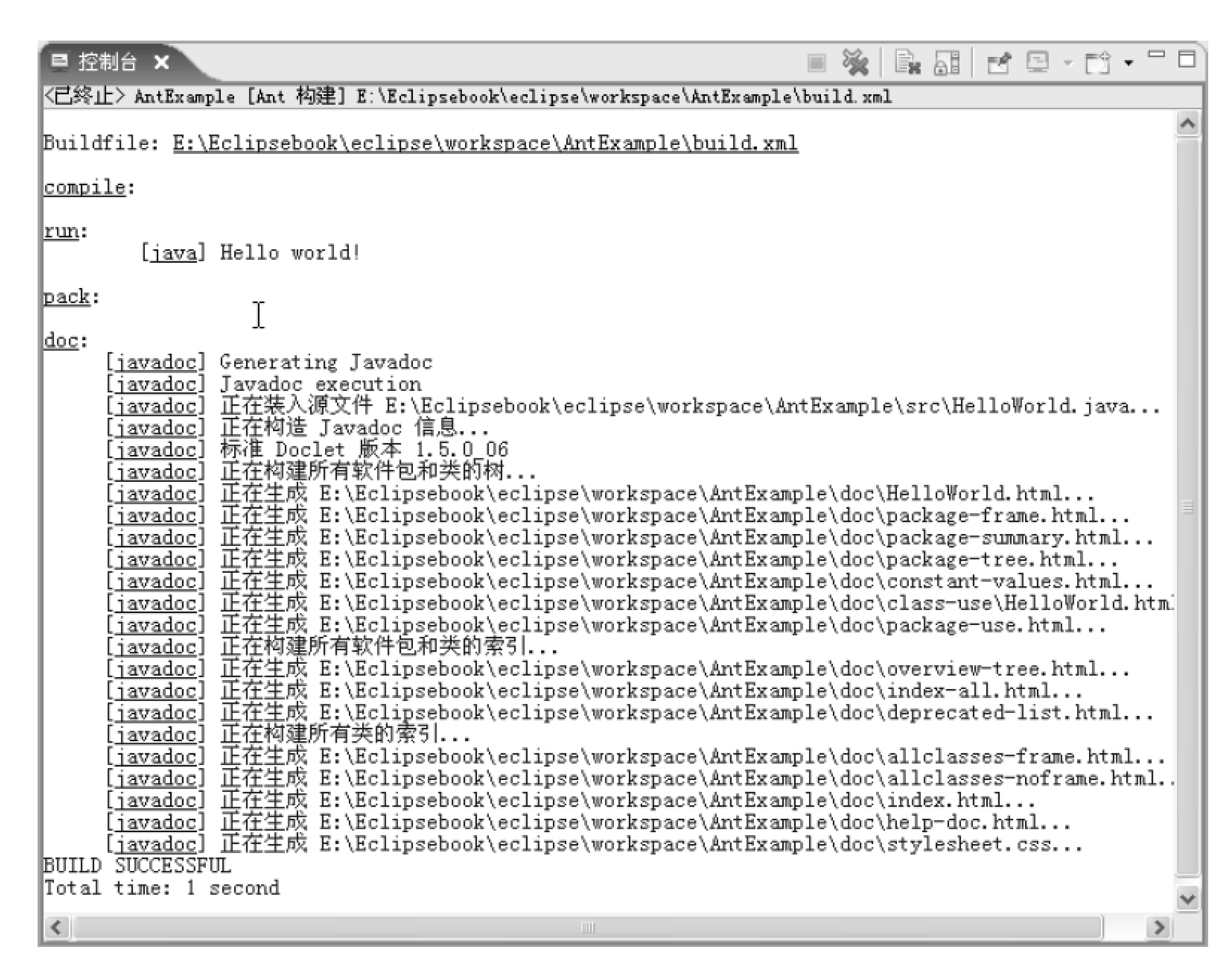


图 4-26　自动构建执行 doc target 的控制台输出信息

4.6　开发自己的 Task（任务）

Ant 的任务大体上可以分为核心任务、可选任务和第三方任务。Ant 的任务具有很强的可扩展性，提供了简单易用的任务扩展接口。本节将介绍如何开发自己的 Task。

4.6.1　建立构建环境

本小节依然选择 Ant 作为构建工具。建立名字为 MyTask 的 Java 工程。利用 Ant 完成如下操作：编译所有的 Java 类、打包成 jar 文件、清除文件。

跟我做

（1）创建名字为“MyTask”的 Java 工程，并创建名字为“src”的源文件夹。

（2）在“MyTask”工程中创建“build.xml”文件。

（3）编辑 build.xml 文件，输入如下代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="jar">
    <!--src 目录用来存放所有的 java 源程序-->
    <property name="src.dir" value="src"/>
    <!--classes 目录用来存放编译后的 classes 程序-->
    <property name="classes.dir" value="classes"/>
    <!--创建名字为 clean 的目标，删除所有产生的文件-->
    <target name="clean" description="Delete all generated files">
```

```
    <!--删除 classes 文件夹-->
        <delete dir="${classes.dir}" failonerror="false"/>
        <!--删除打包后的 jar 文件-->
        <delete file="${ant.project.name}.jar"/>
    </target>
     <!--创建名字为 compile 的目标，编译 src 文件夹下的所有 java 程序-->
    <target name="compile" depends="clean" description="Compiles the Task" >
    <!--创建 classes 文件夹-->
        <mkdir dir="${classes.dir}"/>
        <!--编译 src 文件夹下的所有文件，将编译后的 class 放到 classes 目录下-->
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>
     <!--创建名字为 jar 的目标，将 classes 目录下的 class 文件打包成 jar 文件-->
    <target name="jar" description="JARs the Task" depends="compile">
        <jar destfile="${ant.project.name}.jar" basedir="${classes.dir}"/>
    </target>
</project>
```

4.6.2　第一个简单的 Task

本小节通过一个最简单的 hello World Task，介绍开发第三方任务的基本过程。

跟我做

（1）在“MyTask”工程的“src”源文件夹下建立“HelloWorld.java”文件。

（2）编辑 HelloWorld.java 文件，输入如下代码：

```
public class HelloWorld {
    public void execute() {
        System.out.println("Hello World");
    }
}
```

（3）右击 My Task 的“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台视图输出如图 4-27 所示信息，My Task 的工程树结构如图 4-28 所示。

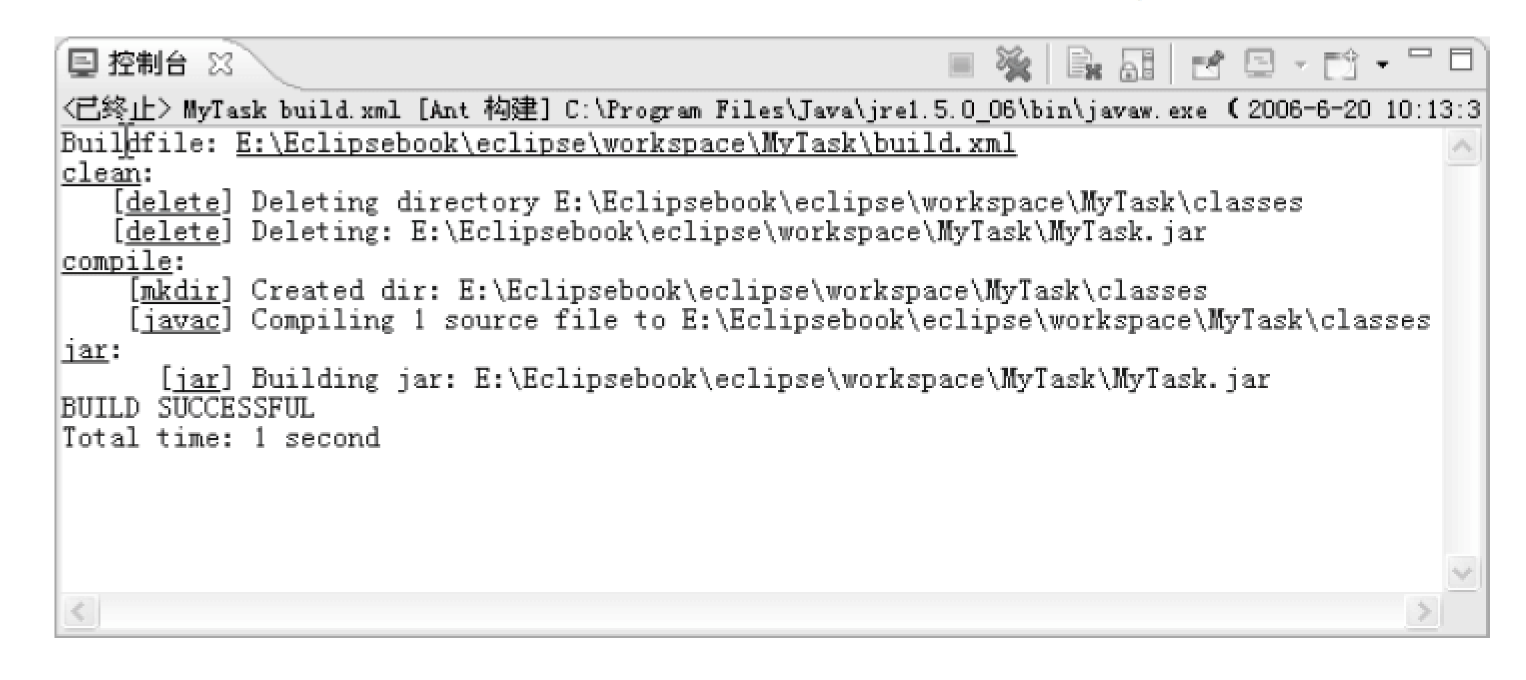


图 4-27　MyTask 构建的输出信息

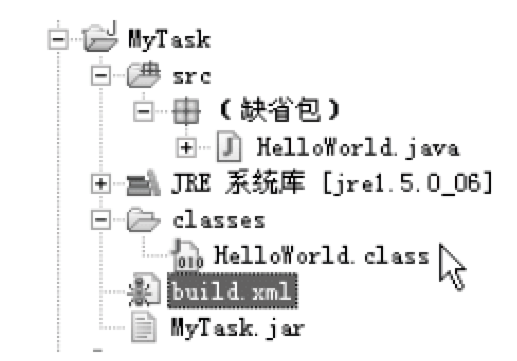


图 4-28　MyTask 工程树结构

（4）在 build.xml 文件中创建名字为“use”的目标，输入如下代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="MyTask" basedir="." default="use">
    ...
    <!--创建名字为 use 的目标-->
    <target name="use" description="Use the Task" depends="jar">
        <!--声明 helloworld 任务-->
        <taskdef name="helloworld"classname="HelloWorld" classpath="${ant.project.name}.jar"/>
        <!--使用 helloworld 任务-->
        <helloworld/>
    </target>
    …
</project>
```

注意： 在使用新定义的任务之前必须使用<taskdef>标签声明。其中 name 属性定义了任务的名称，classname 属性声明了新任务对应的 Class，classpath 属性定义了新任务的 Java 类所在的位置（默认的情况下，Ant 只在其 lib 目录下寻找所需要的 Java 类）。

（5）右击 MyTask 的“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台视图输出如图 4-29 所示信息。

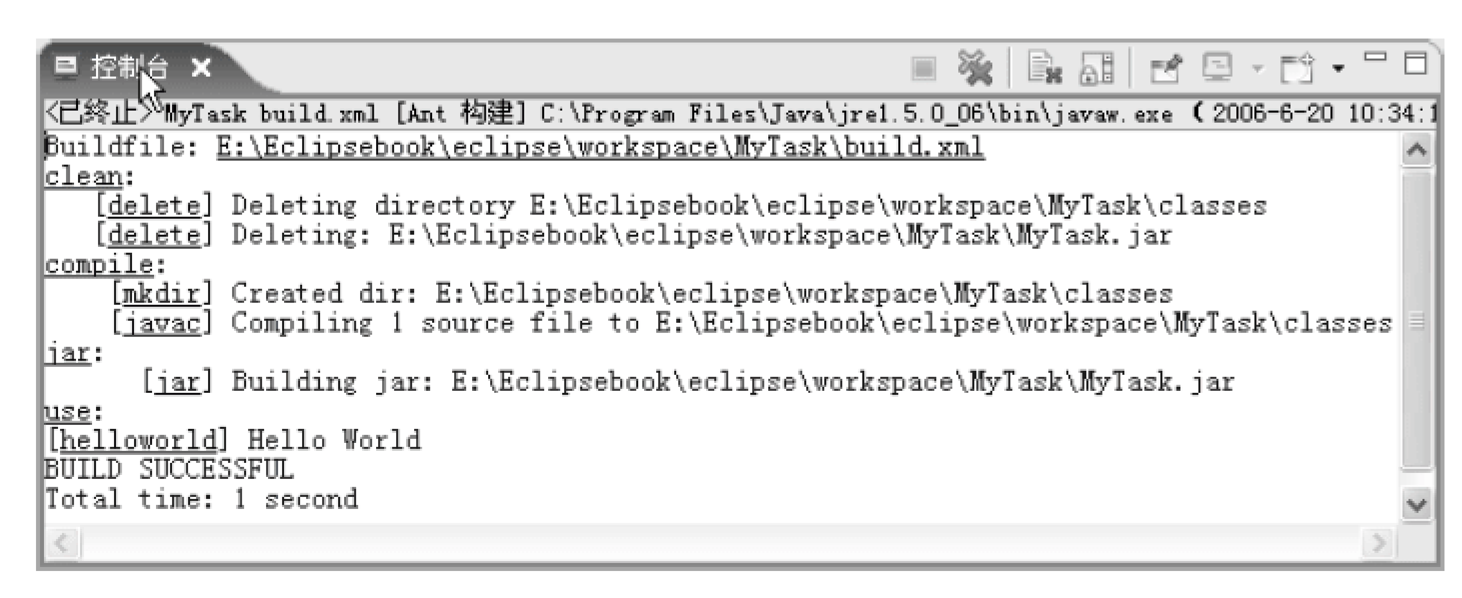


图 4-29　执行 use 目标后的控制台输出信息

4.6.3　开发一个完整的 Task（任务）

本小节介绍一个完整任务的开发过程，该任务具有属性、子元素。新建任务通常的方式是继承 Ant 框架的 org.apache.tools.ant.Task 类，通过该类可以方便地取得 project 的引用，提供文档文件，提供最简单的访问日志的方式。

跟我做

（1）右击“MyTask”工程的“src”源文件夹，在快捷菜单中选择【新建】|【类】命令，打开【新建 Java 类】对话框。

（2）在【名称】文本框中输入“FirstTask”，单击【超类】文本框右侧的【浏览】按钮，打开【超类选择】窗口。

（3）选择“org.apache.tools.ant.Task”类，单击【确定】按钮，返回【新建 Java 类】窗口，如图 4-30 所示。

注意： 在做这一步之前需要将 ant.jar 文件加入到 MyTask 工程的构建路径中。

（4）单击【确定】按钮，创建“FirstTask.java”文件，打开“FirstTask.java”文件，右击该文件的空白区域，在快捷菜单中选择【源代码】|【覆盖/实现方法】命令，打开【覆盖/实现方法】窗口，如图 4-31 所示。



图 4-30　新建 FirstTask 类

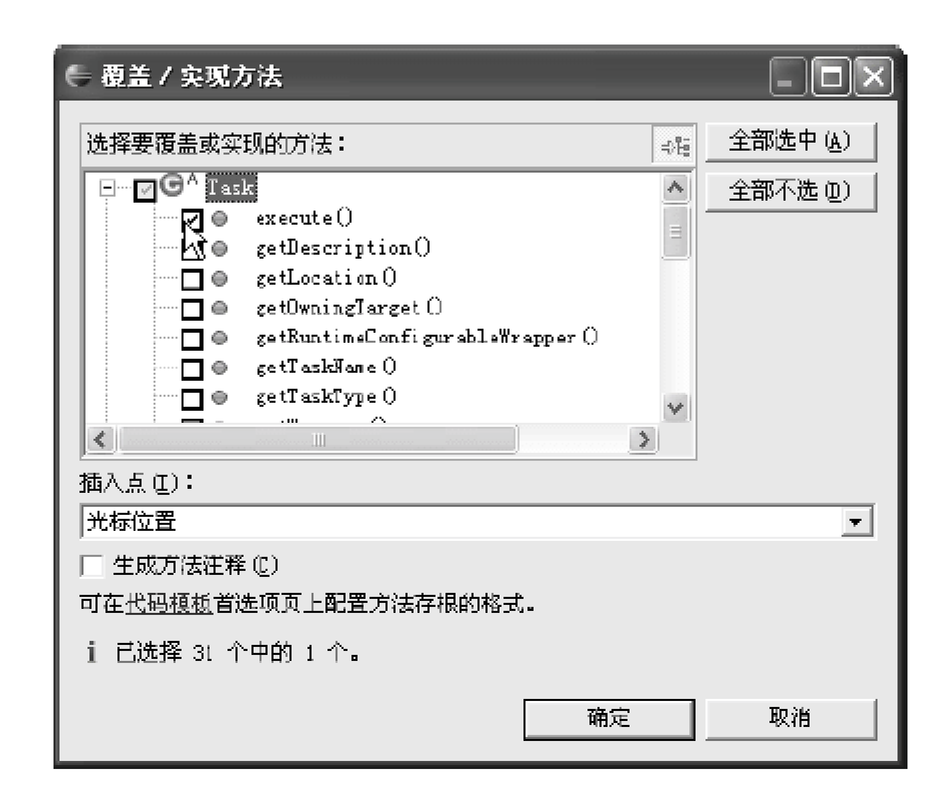


图 4-31　【覆盖/实现方法】窗口

（5）选中“execute()”方法，单击【确定】按钮，在 FirstTask.java 文件中生成如下方法体：

```
public void execute() throws BuildException {

}
```

（6）编辑 FirstTask.java，输入如下程序代码：

```
import org.apache.tools.ant.Task;
import org.apache.tools.ant.BuildException;
import java.util.Vector;
import java.util.Iterator;

//定义自己的 Task，继承 Ant 的 Task
public class FirstTask   extends Task {
    /**  定义名字为 message 的属性*/
    String message;
    //message 属性的 setter 方法
    public void setMessage(String msg) {
        message = msg;
    }

    /**定义布尔型的 fail 属性  */
    boolean fail = false;
    //fail 属性的 setter 方法
```

```
    public void setFail(boolean b) {
        fail = b;
    }

    /**  设置任务支持 Nested 文本*/
    public void addText(String text) {
        message = text;
    }
    //Task 的关键方法，实现特定的功能
    public void execute() {
        //处理属性 fail
        if (fail) throw new BuildException("Fail requested.");
        // 处理属性 message 和 nested 文本
        if (message!=null) log(message);
        // 处理任务的 nested 元素
        for (Iterator it=messages.iterator(); it.hasNext(); ) {
            Message msg = (Message)it.next();
            log(msg.getMsg());
        }
    }
    /**通过一个链表存放 Message 类的多个实例*/
    Vector messages = new Vector();

    /**产生 nested 元素的工厂方法，在 messages 链表中保存 Message 类的引用，并把该引用返
回给 Ant 的核心*/
    public Message createMessage() {
        Message msg = new Message();
        messages.add(msg);
        return msg;
    }

    /**创建一个 Message 类收集子元素包含的所有信息，msg 属性的创建方式跟任务属性的定义方
式是一致的，必须指定之前的 setter 方法 */
    public class Message {
        //Message 的构造函数
        public Message() {}
        /**  打印的 Message */
        String msg;
        public void setMsg(String msg) { this.msg = msg; }
        public String getMsg() { return msg; }
    }
}
```

定义任务中的属性，必须为每个属性写一个 setter 方法。这个 setter 方法是只有一个参数的 public void 方法。方法的名字必须是“set+属性名字（属性名字的第一个字母要大写，其他的为小写）”的形式。如上述所示的 message 属性和 fail 属性，分别对应 setMessage 和 setFail 方法。

Ant 中<echo>任务的使用方法是<echo>Hello World</echo>。定义任务的“Nested Text”

功能需要提供一个 public void addText(String text)方法，将参数的值赋给 message 属性。

定义任务的子元素，首先创建一个 Message 类收集子元素包含的所有信息，然后声明一个工厂方法实例化 build.xml 中定义的所有子元素，并将这些子元素的引用传递给 Ant。

（7）编辑 build.xml 文件，输入如下代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<project name="FirstTask" basedir="." default="use">
    <property name="src.dir" value="src"/>
    <property name="classes.dir" value="classes"/>
    <!--创建名字为 clean 的目标，删除上次构建生成的文件-->
    <target name="clean" description="Delete all generated files">
        <delete dir="${classes.dir}" failonerror="false"/>
        <delete file="${ant.project.name}.jar"/>
    </target>
    <!--创建名字为 compile 的目标，建立 classes 目录，并将 src 文件夹中的所有文件编译-->
    <target name="compile" description="Compiles the Task">
        <mkdir dir="${classes.dir}"/>
        <javac srcdir="${src.dir}" destdir="${classes.dir}"/>
    </target>
     <!--创建名字为 jar 的目标，将 classes 中的文件打包成 jar 文件-->
    <target name="jar" description="JARs the Task" depends="compile">
        <jar destfile="${ant.project.name}.jar" basedir="${classes.dir}"/>
    </target>
    <!--创建名字为 use.init 的目标，定义 helloworld 任务-->
    <target name="use.init"
            description="Taskdef the HelloWorld-Task"
            depends="jar">
        <!--定义 helloworld 任务，类名为 FirstTask-->
        <taskdef name="helloworld"
                 classname="FirstTask"
                 classpath="${ant.project.name}.jar"/>
    </target>
    <!--创建名字为 use.without 的目标，无属性，无包含文本，无子元素-->
    <target name="use.without"
            description="Use without any"
            depends="use.init">
        <helloworld/>
    </target>
     <!--创建名字为 use.message 的目标，具有 message 属性-->
    <target name="use.message"
            description="Use with attribute 'message'"
            depends="use.init">
        <!--helloworld 任务，具有 message 属性值-->
        <helloworld message="attribute-text"/>
    </target>
     <!--创建名字为 use.fail 的目标，具有 fail 属性-->
    <target name="use.fail"
            description="Use with attribute 'fail'"
            depends="use.init">
```

```
            <!--helloworld 任务，具有 fail 属性值-->
            <helloworld fail="true"/>
        </target>
        <!--创建名字为 use.nestedText 的目标，具有包含文本功能-->
        <target name="use.nestedText"
                description="Use with nested text"
                depends="use.init">
             <!--helloworld 任务，具有包含文本-->
            <helloworld>nested-text</helloworld>
        </target>
         <!--创建名字为 use.nestedElement 的目标，具有 message 子元素-->
        <target name="use.nestedElement"
                description="Use with nested 'message'"
                depends="use.init">
            <helloworld>
                <!--message 子元素，具有 msg 属性-->
                <message msg="Nested Element 1"/>
                <message msg="Nested Element 2"/>
            </helloworld>
        </target>
        <!--创建名字为 use 的目标-->
        <target name="use"
                description="Try all (w/out use.fail)"
                depends="use.without,use.message,use.nestedText,use.nestedElement"
        />
</project>
```

（8）右击 MyTask 的“build.xml”文件，在快捷菜单中选择【运行方式】|【Ant 构建】命令，控制台视图输出如图 4-32 所示信息。

（9）打开 Ant 视图，将 MyTask 工程中的 build.xml 拖动到 Ant 视图中，如图 4-33 所示。Ant 视图将 build.xml 中的所有任务以树形的结构组织起来，在其中可以选中任何一个目标，单击按钮来运行该目标。



图 4-32　添加 message 属性后运行 use 目标后的输出信息

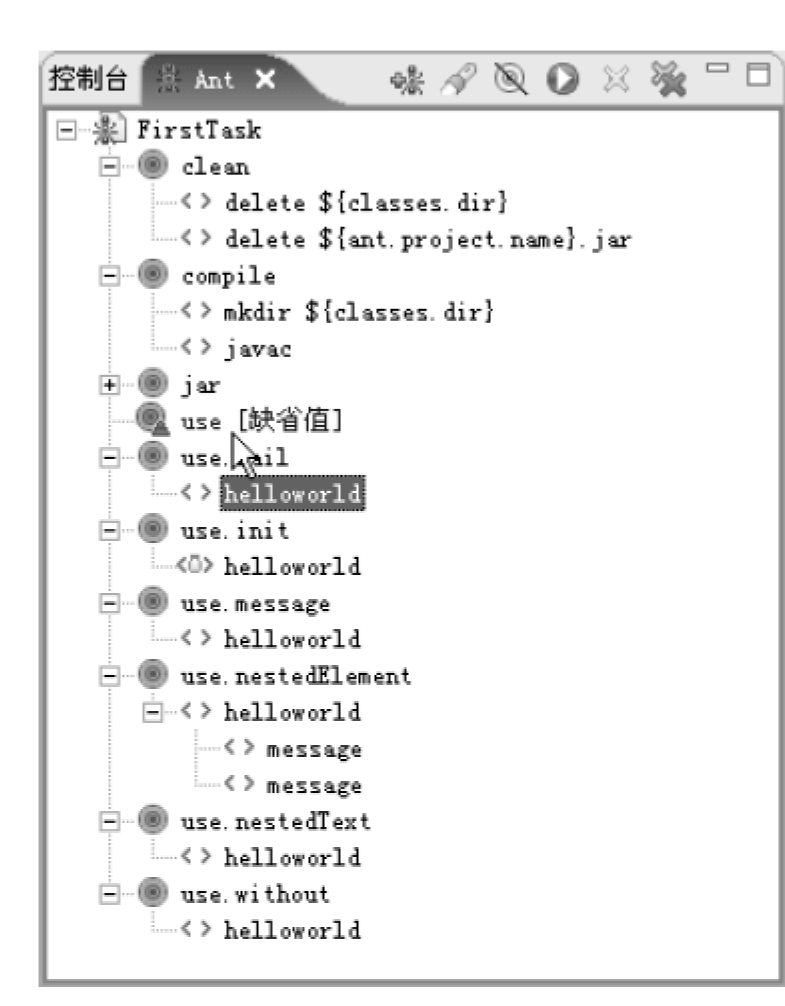


图 4-33　build.xml 文件的 Ant 视图

第 5 章　数据库开发实例——学生成绩管理系统

HSQLDB 是一个开源的纯 Java 嵌入式关系数据库管理系统，小巧方便，具有标准的 SQL 语法和 Java 接口，可以作为内存数据库、独立数据库和 C/S 数据库，支持索引、事务处理、Java 存储过程、完整性引用和约束等功能。

本章介绍 Eclipse 环境下的 HSQLDB 数据库应用开发，包括 HSQLDB 数据库的安装和配置、SqlExplorer 数据库插件的安装和配置、常见数据库操作的封装，最后通过学生成绩管理系统介绍了基于 HSQLDB 进行数据库应用开发的具体步骤。

5.1　HSQLDB 数据库的使用

5.1.1　下载并安装 HSQLDB 数据库

在使用 HSQLDB 数据库之前，本小节首先介绍 HSQLDB 数据库的下载和安装。与大多数 Java 应用程序一样，只需解压缩安装包即可完成 HSQLDB 数据库的安装。

跟我做

（1）登录 HSQLDB 的官方网站 http://www.hsqldb.org，下载 HSQLDB 数据库的安装包 hsqldb_1_8_0_x.zip。

（2）将下载的安装包解压缩到设定的安装目录，如 d:\hsqldb。

（3）将 D:\hsqldb\lib 目录下的 hsqldb.jar 文件加入到 CLASSPATH 环境变量中，HSQLDB 安装完毕。安装后其目录结构如图 5-1 所示。

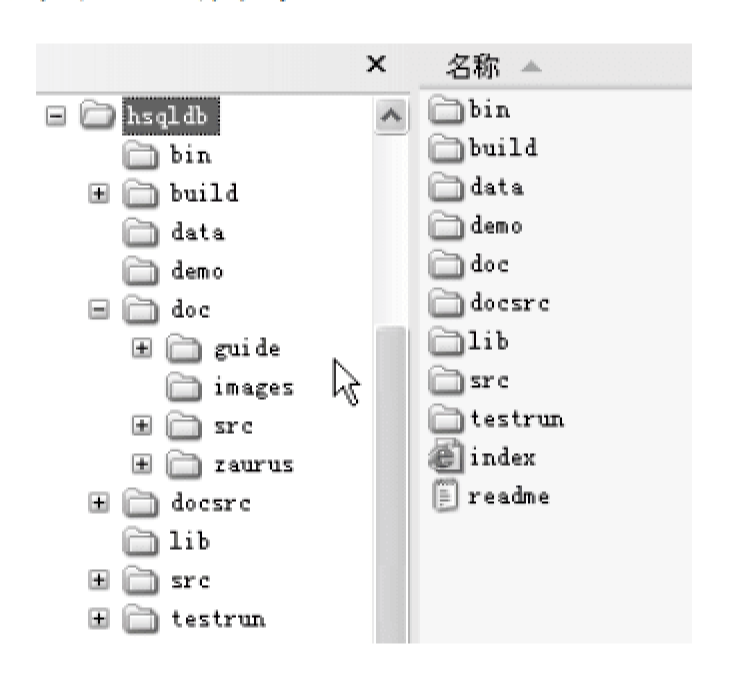


图 5-1　HSQLDB 数据库的目录结构

注意： 设置 CLASSPATH 环境变量的方法参见 1.1 节，所有的 HSQLDB 组件如数据库引擎、服务器进程、JDBC 驱动程序、文档以及一些实用工具都放在 hsqldb.jar 文件中。

5.1.2　使用 Memory 模式运行 HSQLDB

下面介绍 HSQLDB 的几种运行模式。

- 独立服务器模式：类似于其他关系数据库的标准客户机/服务器数据库配置，允许出现使用 TCP 套接字的并发连接。
- 独立 Web 服务器模式：作为 Web 服务器通过 HTTP 接受 SQL 查询，也能作为任何标准 Web 容器中的 Servlet 来运行。由于 HTTP 是无状态的，所以本模式中不存在事务。
- 单机模式：是许多嵌入式应用程序的首选模式，该模式下应用程序使用 JDBC 创建一个数据库连接，HSQLDB 引擎也运行在该应用程序中。
- Memory 模式：所有数据库表和索引都放在内存中，数据不进行外存储，没有持久性。

本小节将以 Memory 模式为例，介绍如何基于 HSQLDB 数据库进行应用的开发。

跟我做

（1）启动 Eclipse，创建名字为 hsqldbdemo 的 Java 工程，并创建 Java 类 MemoryDB.java。

注意： 切记将 hsqldb.jar 加到工程的构建路径上。

（2）编辑 MemoryDB.java 文件。输入如下代码：

```
try {
		//加载 HSQLDB 数据库 JDBC 驱动
		Class.forName("org.hsqldb.jdbcDriver");
		//在内存中建立临时数据库 score，用户名为 sa，密码为空
		Connection connect = DriverManager.getConnection("jdbc:hsqldb:mem:score",
				"sa", "");
		System.out.println("在此行上设置一个断点");
	} catch (SQLException e) {
		e.printStackTrace();
	} catch (ClassNotFoundException e) {
		e.printStackTrace();
	}
```

在内存中建立临时数据库“score”，用户名为“sa”，密码为空。上述程序片断是典型的通过 JDBC 连接数据库的方法，其中数据库 URL“jdbc:hsqldb:mem:score”中的“mem”部分定义了 HSQLDB 数据库工作在 Memory 模式下。一旦跟数据库的连接建立后，数据库引擎就启动起来了，接下来即可创建 Table 表。

5.2　使用 SQLExplorer 插件连接数据库

SQLExplorer 插件可以通过 JDBC 访问常用的关系数据库，同时也支持像 Hibernate 这样的工具访问数据库。其官方站点为 http://sourceforge.net/projects/eclipsesql。

本节介绍如何使用 SQLExplorer 插件，查看 5.1.2 小节建立的 score 内存数据库的具体内容。首先介绍 SQLExplorer 插件的安装，然后介绍 SQLExplorer 插件的具体使用方法。

工作在内存模式下的 HSQLDB 数据库，会随着程序的退出而关闭，所以在下面的操作中 MemoryDB 要始终保持运行状态。本章在程序中设置断点，调试运行，使程序保持运行状态。

跟我做

（1）打开 MemoryDB.java 文件，在程序行“System.out.println（“在此行上设置一个断点”）;”前设置一个断点。

（2）右击“MemoryDB.java”文件，在快捷菜单中选择【调试方式】|【Java 应用程序】命令，MemoryDB.java 程序调试运行至断点处，建立了内存数据库 score。

（3）单击【窗口】菜单，依次选择【打开透视图】|【其它...】命令，打开【选择透视图】对话框，选择“SQLExplorer”，打开 SQLExplorer 透视图，如图 5-2 所示。



图 5-2　选择 SQLExplorer 透视图

SQLExplorer 透视图有 7 项内容，分别如下：

- Aliases 别名，用来标识数据库连接串。
- Connection Info 连接信息，用来显示连接数据库时的相关信息，如数据库产品名称、版本、JDBC 驱动程序的名称、版本、用户名、连接串、是否自动提交等。
- Connections 显示活动的连接情况。
- Database Structure View 显示数据库结构。

- ❑ Drivers 配置驱动程序用。
- ❑ SQL History 执行 SQL 的历史记录。
- ❑ SQL Results 执行 SQL 的结果集。

（4）打开如图 5-3 所示的 Drivers 视图，右击“HSQLDB In-Memory”，在快捷菜单中选择【Change the selected Driver】命令，打开 Modify Driver 窗口。

（5）选择【Extra Class Path】选项卡，单击【Add】按钮，在【打开】窗口中选择 d:\hsqldb\lib\hsqldb.jar，将 HSQLDB 数据库的驱动程序加入到 classpath 中。

（6）在【Example URL】文本框中输入“jdbc:hsqldb:mem:score”，单击【确定】按钮，如图 5-4 所示。这时，Drivers 视图中的“HSQLDB In-Memory”由⊗变成✓，表示 HSQLDB 数据库的驱动程序配置成功。



图 5-3　Drivers 视图

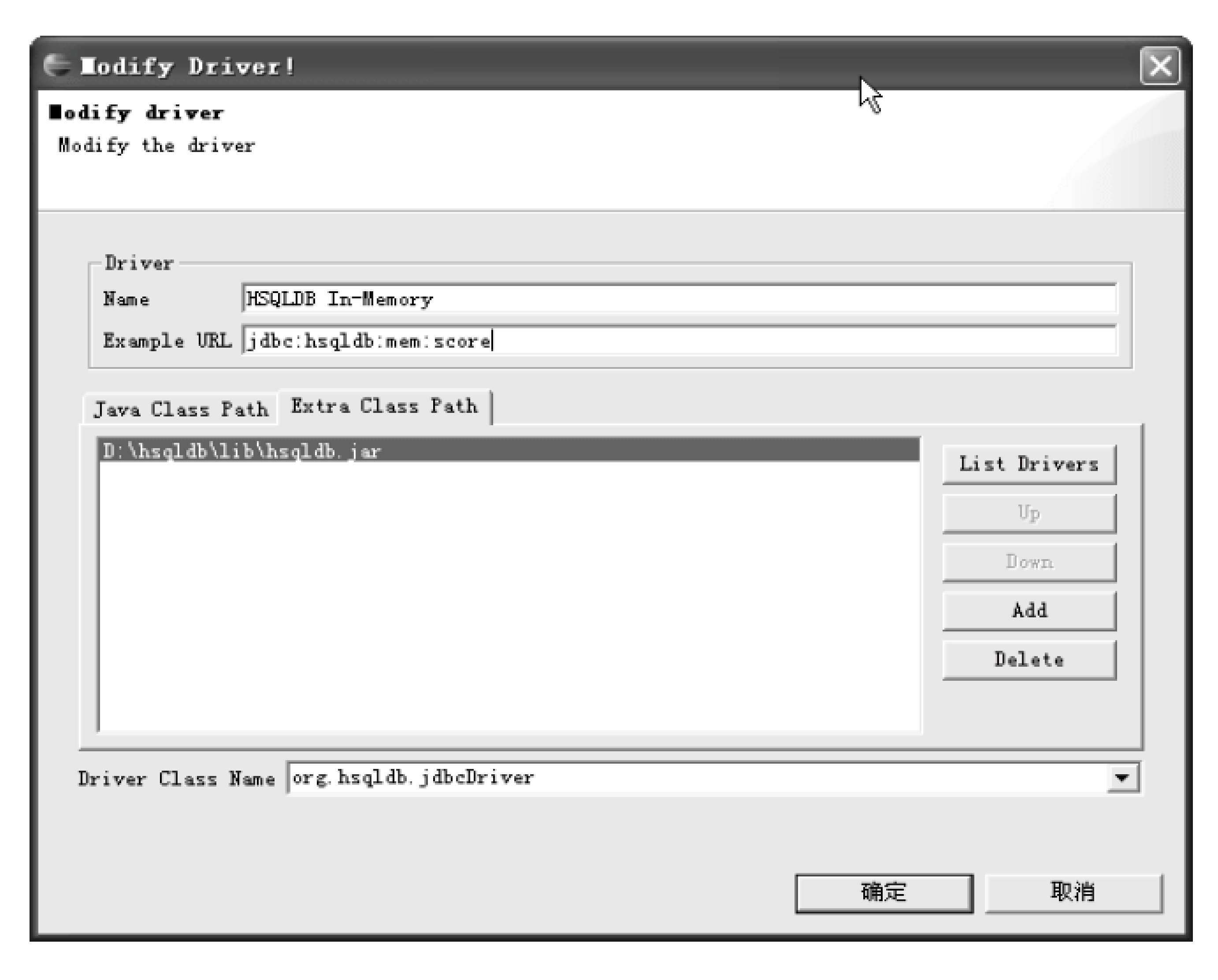


图 5-4　配置数据库驱动

（7）打开 SQLExplorer 插件的 Aliases 别名视图，单击【创建】图标，打开【Create new Alias】对话框。

（8）在【Name】文本框中输入“hsqlMemoryDB”，选择 HSQLDB In-Memory 驱动，在【URL】文本框中输入“jdbc:hsqldb:mem:score”，在【User Name】文本框中输入“sa”，单击【确定】按钮，如图 5-5 所示。在 Aliases 别名视图中出现刚建立的“hsqlMemoryDB”连接。

（9）右击“hsqlMemoryDB”，在快捷菜单中选择【Open...】命令，弹出有关数据库连接的确认框，可以更改用户名与密码，也可以设置是否自动提交，这里保持所有的选项为默认值。

（10）单击【确定】按钮，在 Database Structure View 视图中即可看到 Database，展开 Database 树形结构，如图 5-6 所示。

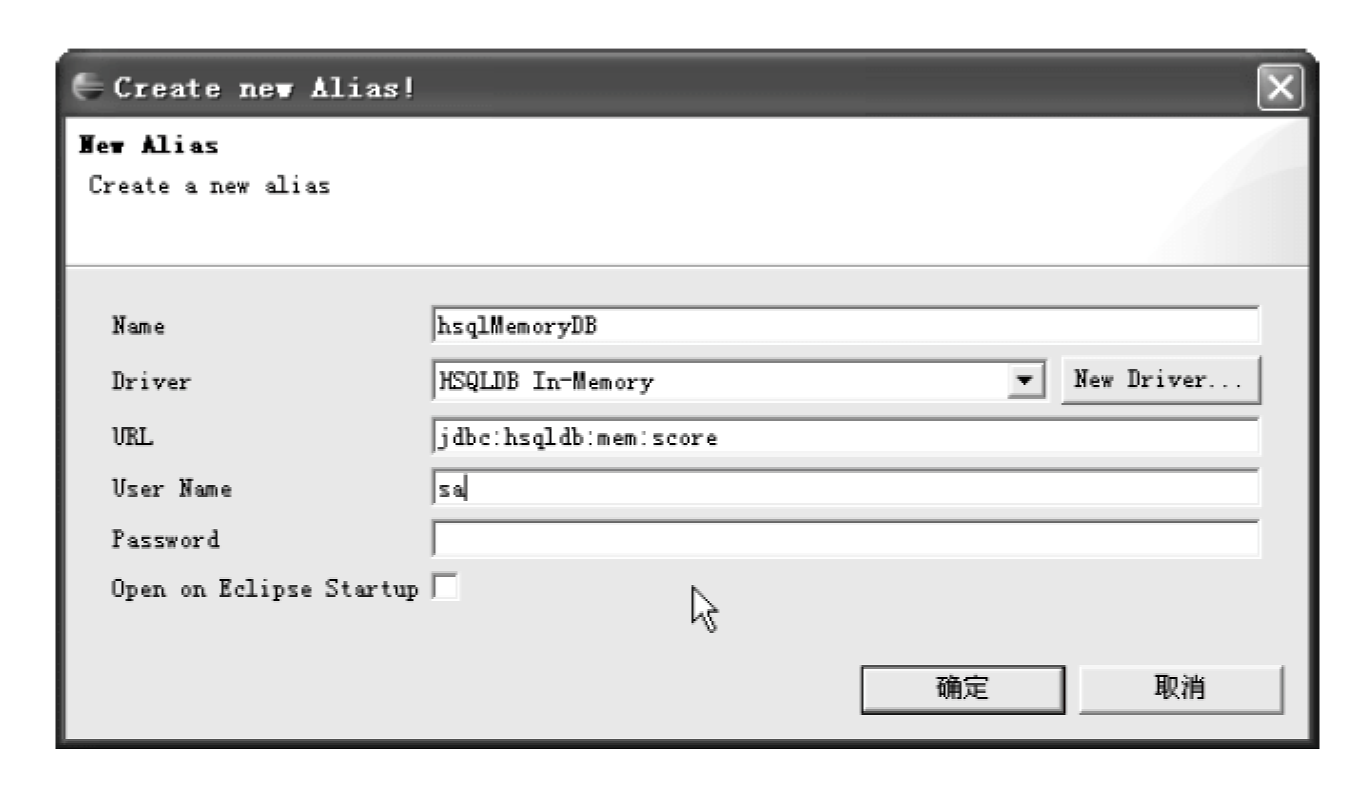


图 5-5　创建数据库连接别名

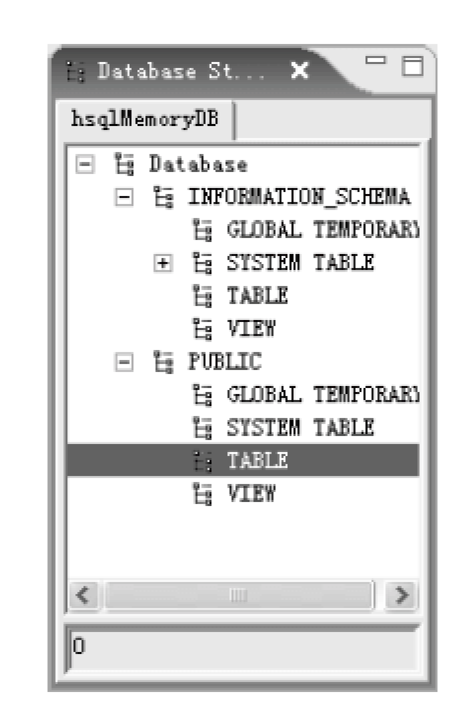


图 5-6　Database Structure View 视图

5.3　创建 Score 成绩表

至此通过 SQLExplorer 插件已经建立了与 HSQLDB 数据库的连接。本节通过在 SQLExplorer 中编辑和运行 SQL 脚本建立 Score 表，为后续几节的数据库操作做准备。

5.3.1　编写脚本

跟我做

（1）打开 SQLExplorer 插件的 Connections 视图，其中显示当前数据库的连接情况，这里有一个活动的连接，如图 5-7 所示。

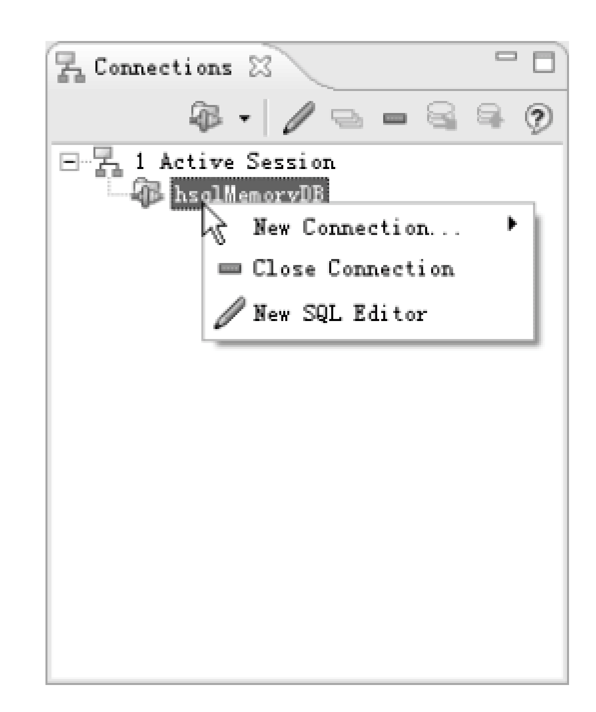


图 5-7　Connections 视图

（2）单击 Connections 视图中的按钮，选择【New SQL Editor】命令，创建一个新的 SQL 编辑器。

（3）在 SQL 编辑器中输入如下 SQL 语句。

```
CREATE TABLE Score
  (SNO CHAR(7)   NOT NULL,
  CNO CHAR(6) NOT NULL,
  GRADE NUMERIC(4,1),
  PRIMARY KEY(SNO,CNO));
```

该 SQL 语句在 score 内存数据库中创建一张名为 Score 的表。该表包括 3 个字段：学号（SNO）、课程编号（CNO）、课程分数（GRADE），并且以 SNO、CNO 两个字段作为主键。

5.3.2　在 SQLExplorer 中运行脚本

SQL Editor 是 SQLExplorer 插件提供的功能强大、使用方便的 SQL 语句编辑器，如图 5-8 所示。SQL Editor 提供编辑 SQL 语句，选中并执行部分 SQL 语句，打开和保存 SQL 脚本文件等功能。本小节介绍如何在 SQLExplorer 中运行 SQL 语句。

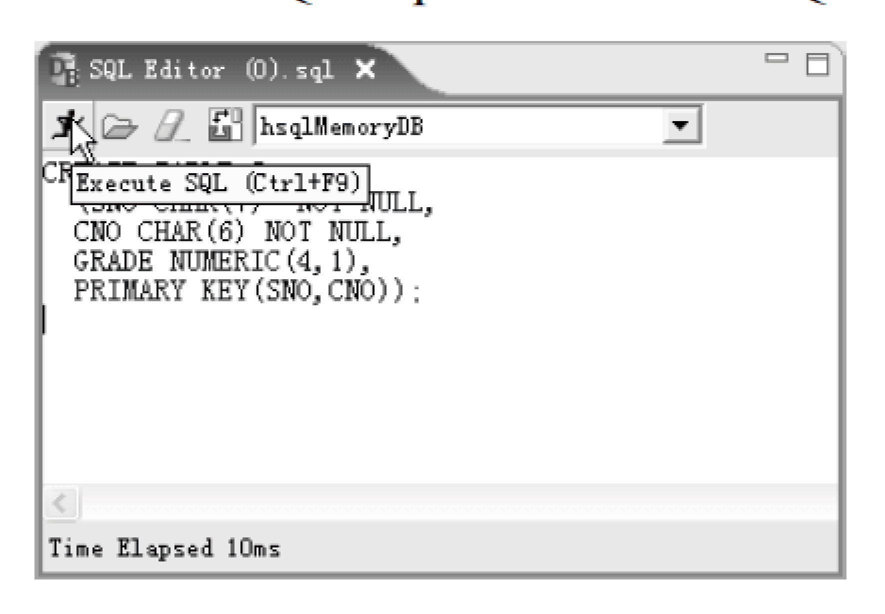


图 5-8　SQL Editor

跟我做

（1）单击【Execute SQL】按钮，执行所输入的 SQL 语句。

注意： 如果 SQL Editor 中存在多条 SQL 语句，首先选中想要执行的语句，然后单击【Execute SQL】按钮，如图 5-9 所示。

（2）从打开 SQLExplorer 插件的 Database Structure View 视图中可以看到刚才创建的表格。选中该表，可以看到该表的详细结构，如图 5-10 所示。

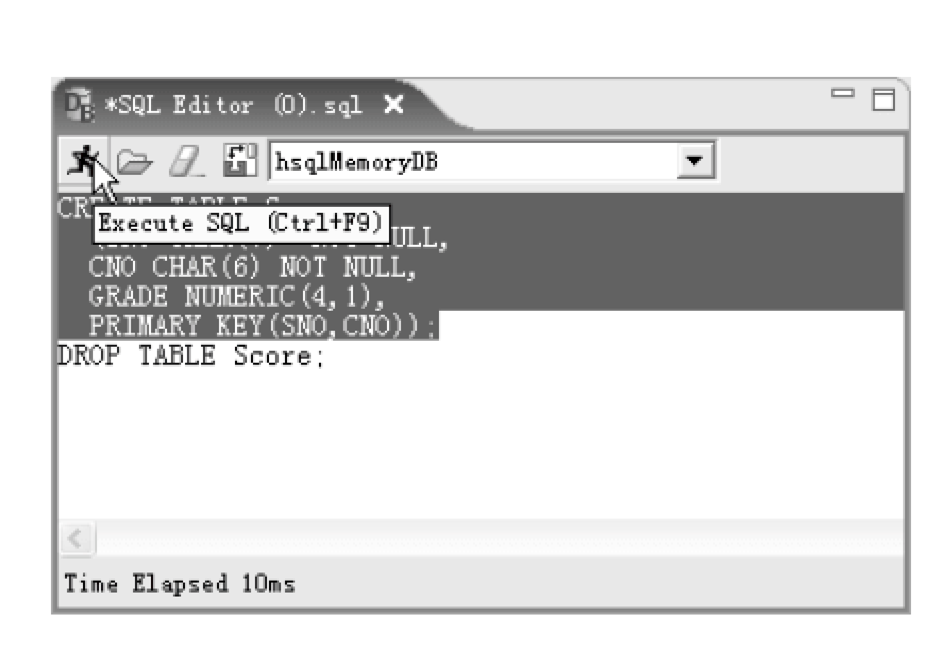


图 5-9　执行多条 SQL 语句中的某一条

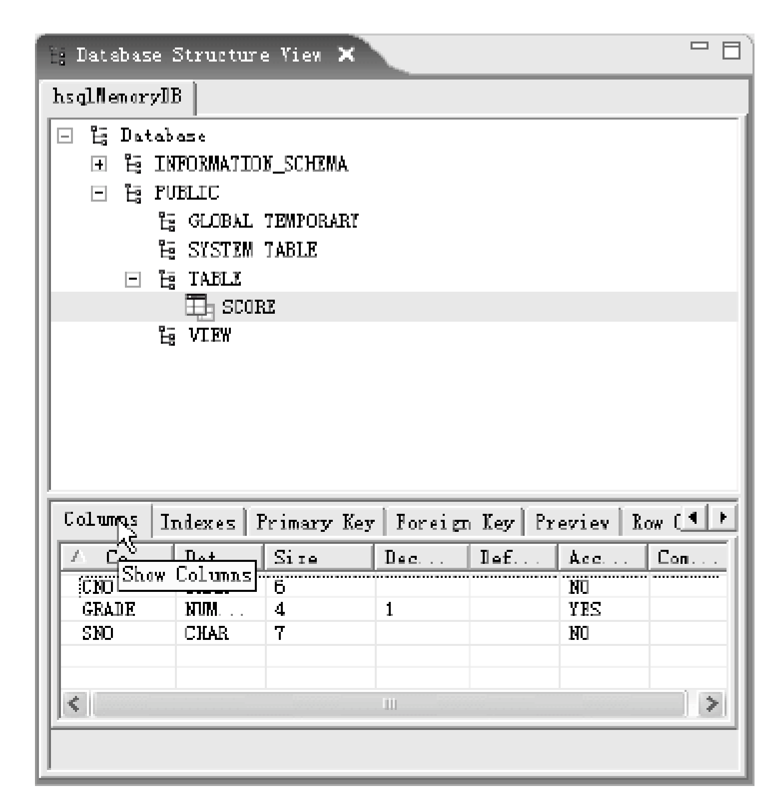


图 5-10　新创建的 Score 表

（3）打开 SQLExplorer 插件的 SQL History 视图，可以看到执行过的所有 SQL 语句列表，如图 5-11 所示。

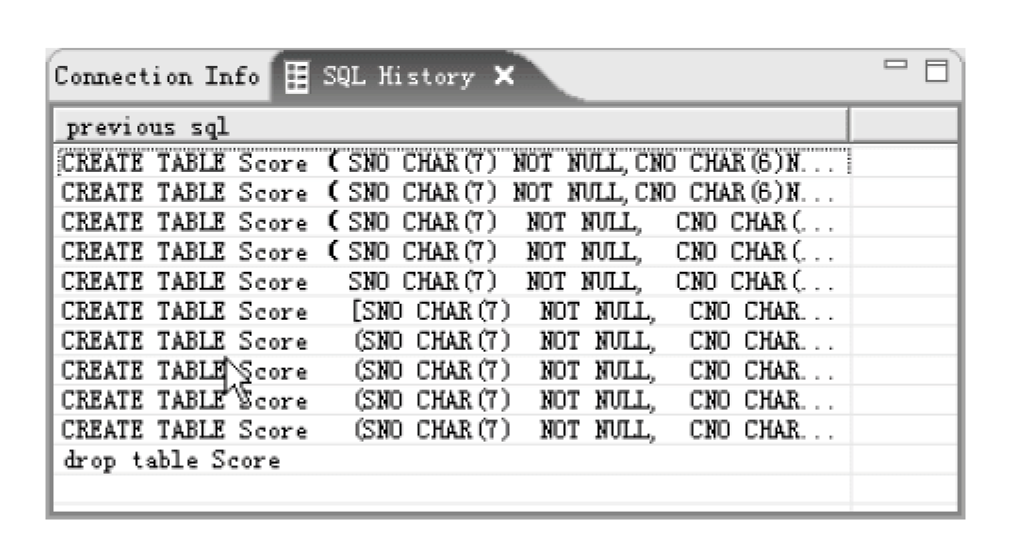


图 5-11　SQL History 视图

5.4　使用 JavaBean 映射成绩表

对象关系映射（ORM）是一种为了解决面向对象与关系数据库存在的互不匹配的现象的技术。其本质就是将数据从关系数据库的二维表格形式改换一种形式，以更加容易理解的面向对象形式来表现具有某种关系的数据。本节将遵循 ORM 的思想，以 Score 表为例说明如何以 JavaBean 的方式来映射 HSQLDB 数据库中的 Score 表。

5.4.1　实现 Score 类

表 Score 共包括 3 个字段：SNO（学号）、CNO（课程编号）、GRADE（分数）。下面创建一个 JavaBean 来映射这个表。

跟我做

（1）右击“hsqldbdemo”工程的“src”源文件夹，在快捷菜单中选择【新建】|【包】命令，打开【新建 Java 包】窗口。

（2）在【名称】文本框中输入“hsqldb.javabean”，单击【完成】按钮，建立名为“hsqldb.javabean”的 package。

（3）在包“hsqldb.javabean”中创建 Java 类 Score.java。

（4）编辑 Score.java 文件，输入如下代码：

```
package hsqldb.javabean;
//映射 HSQLDB 数据库 Score 表的 JavaBean 类
public class Score {
	// 学号
	private String SNO;
	// 课程编号
	private String CNO;
	// 课程分数
	private float GRADE;
}
```

Score 类为标准的 JavaBean，为 Score 类声明了 3 个属性：SNO、CNO、GRADE，分别表示学号、课程编号和课程分数。

5.4.2　添加 getter/setter 方法

为了能够实现查询、修改等数据库常见操作，需要为 Score 类加入 getter/setter 方法。

跟我做

（1）打开 Score.java 文件，右击文件中的任何空白区域，在快捷菜单中选择【源代码】|【生成 Getter 和 Setter】命令，打开【生成 Getter 和 Setter】窗口，如图 5-12 所示。

（2）单击【全部选中】按钮，全部选中 3 个属性，单击【确定】按钮。生成完整 Score 类代码如下：

```
package hsqldb.javabean;
//映射 Score 表的 Javabean 类
public class Score {
	// 学号
	private String SNO;
	// 课程编号
	private String CNO;
	// 课程分数
	private float GRADE;
	 //CNO 属性的 getter 方法
	public String getCNO() {
		return CNO;
	}
	 //CNO 属性的 setter 方法
	public void setCNO(String cno) {
		CNO = cno;
	}
	//GRADE 属性的 getter 方法
	public float getGRADE() {
		return GRADE;
	}
	//GRADE 属性的 setter 方法
	public void setGRADE(float grade) {
		GRADE = grade;
	}
	//SNO 属性的 getter 方法
	public String getSNO() {
		return SNO;
	}
	//SNO 属性的 setter 方法
	public void setSNO(String sno) {
		SNO = sno;
	}
}
```



图 5-12　【生成 Getter 和 Setter】窗口

为 Score 类的 3 个属性添加 Getter/Setter 方法。Score 类为标准的 JavaBean。

5.5　使用 ScoreDAO 管理成绩

DAO 是数据访问接口 Data Access Object 的简称，在业务逻辑与数据库资源之间。DAO 是一种常用的设计模式，可以用来封装数据库的驱动、数据库 URL、用户名和密码。以后要更改数据库的类型，如把 MSSQL 换成 Oracle，则只需要更改 DAOFactory 里面的相关信息即可。另外，DAO 把对数据库的操作如最基本的查询、更新、删除和插入全部封装在里面，如加入一个学生一门课的成绩，只要调用 DAO 中的 insertScore（Score score）方法即可，省去了编写复杂的 SQL 语句的麻烦，使得操作数据库的动作更加方便。

跟我做

（1）建立 ScoreDAOFactory 类。通过该工厂类建立了和数据库的连接，可以关闭与数据库的连接，另外通过该工厂类可以取得一个 ScoreDAO 的实例。代码如下：

```
package hsqldb.dbo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
//创建 ScoreDAO 类的工厂类
public class ScoreDAOFactory {
	//HSQLDB 数据库的 Driver 名称
	public static final String DRIVER = "org.hsqldb.jdbcDriver";
	//将建立的内存数据库的 URL
	public static final String URL = "jdbc:hsqldb:mem:score";
	//Connection 对象，表示到数据库的连接
```

```
        private static Connection connection = null;
        /**
         * 建立到内存数据库的连接
         * @return
         */
        public static Connection createConnection() {
                if (connection == null) {
                        try {
                               //加载数据库驱动程序
                                Class.forName(DRIVER);
                               //建立到数据库的连接
                                connection = DriverManager.getConnection(URL, "sa", "");
                        } catch (SQLException e) {
                                e.printStackTrace();
                        } catch (ClassNotFoundException e) {
                                e.printStackTrace();
                        }
                }
                return connection;
        }
        /**
         * 释放到内存数据库的连接
         */
        public static void closeConnection() {
                if (connection != null)
                        try {
                                //释放到数据库的连接
                                connection.close();
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
        }
        /**
         * 取得一个 ScoreDAO 的实例
         * @return
         */
        public static ScoreDAO getScoreDAO() {
                return new ScoreDAO();
        }
}
```

该类为 ScoreDAO 的工厂类，用来取得一个 ScoreDAO 的实例。

（2）建立 ScoreDAO 类，封装常见的数据库操作。编写代码如下：

```
package hsqldb.dbo;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import hsqldb.javabean.Score;
```

```
//ScoreDAO 类，用来封装对数据库的常见操作
public class ScoreDAO {
	// 数据库的 JDBC 连接
	private Connection connection;
	//  Statement 对象
	private Statement statement;
	// 需要执行的 SQL 语句
	private String sql;
	/**
	 * 构造函数，建立到 score 数据库的连接，同时在 score 数据库中建立 Score 表
	 */
	public ScoreDAO() {
		// 建立到 score 数据库的连接
		connection = ScoreDAOFactory.createConnection();
		try {
			//创建 statement 对象
			statement = connection.createStatement();
			//创建 Score 表的 SQL 语句
			sql = "CREATE TABLE Score (SNO CHAR(7)  NOT NULL,CNO CHAR(6) NOT
NULL,GRADE NUMERIC(4,1),PRIMARY KEY(SNO,CNO))";
			//执行 SQL 语句
			statement.execute(sql);
		} catch (SQLException e) {
			e.printStackTrace();
		}
	}
}
```

首先取得数据库的连接，然后执行“CREATE TABLE”SQL 语句，创建 Score 表。该表具有 3 个字段：学号、课程编号和分数。

5.5.1　添加 InsertScore 方法增加成绩

“INSERT INTO”语句用来向数据库中插入新的记录。为了简化插入记录操作，完全以面向对象的方式实现对数据库的插入操作，本小节介绍如何对插入语句进行封装。

跟我做

封装 INSERT INTO 标准 SQL 语句。编写代码如下：

```
	/**
	 * 将 Score 的一个对象插入到数据库中
	 *
	 * @param score
	 */
	public void insertScore(Score score) {
		//将参数 score 类的各个属性拼接成插入记录的 SQL 语句
		sql = "INSERT INTO SCORE VALUES(" + "\'" + score.getSNO() + "\',"
```

```
                    + "\'" + score.getCNO() + "\'," + score.getGRADE() + ")";
            try {
                //程序 INSERT INTO 语句为 Score 类插入一条记录，记录的三个字段值对应着 score
类的三个属性
                statement.execute(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            }
    }
```

将 Score 类的属性拼接成 INSERT 语句，这样进行数据库的插入操作只需传入一个 Score 类的实例，完全以面向对象的方式往数据库中插入记录。

5.5.2　添加 SelectScore 方法查询成绩

查询操作是数据库的常见操作，这里通过对 SELECT SQL 语句的封装实现对指定记录的查询以及遍历查询两个方法。

跟我做

（1）对指定记录的查询。在建立表格 Score 时，指定了以 SNO 和 CNO 作为该表的主键。该方法实现了查询 SNO 和 CNO 字段值与给定 Score 对象 SNO 和 CNO 属性值相同的记录。代码如下：

```
    /**
     * 在数据库中查询包含某个 Score 对象信息的记录
     *
     * @param score
     * @return
     */
    public ResultSet selectScore(Score score) {
        //查询后的结果集
        ResultSet result = null;
        //将参数 Score 类的各个属性拼接成查询记录的 SQL 语句
        sql = "select * from score where SNO=" + score.getSNO() + "and CNO="
                + score.getCNO();
        try {
            //执行查询，返回与 Score 类的 SNO、CNO 属性值相同的记录
            result = statement.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return result;
    }
```

（2）对所有记录的查询。为了更加方便地查询数据库目前记录情况，这里实现了简单返回所有记录的遍历查询。编写代码如下：

```
/**
     * 查询数据库中的所有记录
     *
     * @return
     */
    public ResultSet selectAll() {
        //查询后的结果集
        ResultSet result = null;
        //查询所有记录的 SQL 语句
        sql = "select * from score";
        try {
            //执行查询，返回所有的记录
            result = statement.executeQuery(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return result;
    }
```

5.5.3　添加 DeleteScore 方法删除成绩

“DELETE”语句用来从数据库中删除记录。为了简化删除记录操作，完全以面向对象的方式实现对数据库的删除操作，本小节介绍如何对 DELETE 语句进行封装。

跟我做

封装 DELETE SQL 语句，删除数据库中 SNO 和 CNO 字段值与指定的 Score 对象的 SNO、CNO 值相同的记录。编写代码如下：

```
    /**
     * 删除某个数据库记录
     *
     * @param score
     */
    public void deleteScore(Score score) {
        //将参数 Score 类的各个属性拼接成删除记录的 SQL 语句
        sql = "delete from score where SNO=" + score.getSNO() + "and CNO="
                + score.getCNO();
        try {
            //执行删除操作的 SQL 语句
            statement.execute(sql);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

5.5.4　添加 UpdateScore 方法修改成绩

“UPDATE”语句用来更新数据库中的某条记录。为了简化更新操作，完全以面向对象的方式实现对数据库的更新，本小节介绍如何对 UPDATE 语句进行封装。

跟我做

封装 UPDATE SQL 语句将数据库中某条记录更新为指定的 Score 类的 SNO、CNO 和 GRADE 值。编写代码如下：

```
/**
 * 更新数据库中的某条记录
 *
 * @param oldScore
 * @param newScore
 */
public void updateScore(Score oldScore, Score newScore) {
	//将数据库中满足条件的记录更新为 newScore 类的 SNO、CNO 和 GRADE 属性值
	sql = "update score set SNO=" + newScore.getSNO() + ",CNO="
			+ newScore.getCNO() + ",GRADE=" + newScore.getGRADE()
			+ " where SNO=" + oldScore.getSNO() + "and CNO="
			+ oldScore.getCNO();
	try {
		//执行数据库更新语句
		statement.executeUpdate(sql);
	} catch (SQLException e) {
		e.printStackTrace();
	}
}
```

5.6　编写测试客户端

采用 DAO 模式的目的就在于简化对数据库的操作。本节将以一个操作数据库的实例来说明以 DAO 方式访问数据库的具体步骤，从而深刻体会 DAO 模式的实用性。

跟我做

（1）通过 ScoreDAOFactory 取得一个 ScoreDAO 的实例，同时在内存中创建了 score 数据库，并且建立了具有 3 个字段的 Score 表。

（2）准备 3 个 Score 类的对象，分别为 firstScore、secondScore 和 thirdScore。编写代码如下：

```
//firstScore 实例，
Score firstScore = new Score();
//CNO 字段值为 34
```

```
firstScore.setCNO("34");
//SNO 字段值为 1
firstScore.setSNO("1");
//GRADE 字段值为 2.5
firstScore.setGRADE((float) 2.5);
// secondScore 实例
Score secondScore = new Score();
//CNO 字段值为 45
secondScore.setCNO("45");
//GRADE 字段值为 67.9
secondScore.setGRADE((float) 67.9);
//SNO 字段值为 2
secondScore.setSNO("2");
// thirdScore 实例
Score thirdScore = new Score();
//CNO 字段值为 78
thirdScore.setCNO("78");
//SNO 字段值为 3
thirdScore.setSNO("3");
//GRADE 字段值为 89
thirdScore.setGRADE((float) 89.0);
```

（3）将 firstScore 插入到数据库中。编写代码如下：

```
// 通过 ScoreDAO 的实例执行插入操作
scoreDAO.insertScore(firstScore);
// 查询数据库中的所有记录
result = scoreDAO.selectAll();
// 输出所有记录信息
info(result);
```

运行结果如图 5-13 所示。

```
学号=1,课程编号=34,分数=2.5
```

图 5-13　插入 firstScore 记录的执行结果

（4）将 secondScore 插入到数据库中。编写代码如下：

```
// 通过 ScoreDAO 的实例执行插入操作
scoreDAO.insertScore(secondScore);
// 查询数据库中的所有记录
result = scoreDAO.selectAll();
// 输出所有记录信息
info(result);
```

运行结果如图 5-14 所示。

```
学号=1,课程编号=34,分数=2.5
学号=2,课程编号=45,分数=67.9
```

图 5-14　插入 secondScore 记录的执行结果

（5）将 secondScore 记录修改为 thirdScore。编写代码如下：

```
//通过 ScoreDAO 的实例执行更新操作
scoreDAO.updateScore(secondScore, thirdScore);
//查询数据库中的所有记录
 result = scoreDAO.selectAll();
  //输出记录信息
 info(result);
```

运行结果如图 5-15 所示。

```
学号=1,课程编号=34,分数=2.5
学号=3,课程编号=78,分数=89.0
```

图 5-15　更新 secondScore 后的执行结果

（6）将 firstScore 记录删除。编写代码如下：

```
//通过 ScoreDAO 的实例执行删除操作
scoreDAO.deleteScore(firstScore);
//查询数据库中的所有记录
 result = scoreDAO.selectAll();
//输出记录信息
 info(result);
```

运行结果如图 5-16 所示。

```
学号=3,课程编号=78,分数=89.0
```

图 5-16　删除 firstScore 后的执行结果

以上简单的实例实现了数据库的查询、插入、删除和更新操作。整个操作完全以面向对象的形式进行，屏蔽了复杂的 SQL 语句，提高了程序的可读性和开发效率。

第 6 章　Web 开发实例——学生成绩管理系统的改进

J2EE 技术是目前使用最广泛的 Web 应用开发技术。JBoss 推出的 Eclipse IDE 开发工具支持 J2EE 的 Web 和 EJB 开发，提供 Ant 和 Xdoclet 自动提示功能，还提供 Hibernate、JBoss AOP 的开发，内置 JSP、HTML 和 XML 编辑器，提供方便快捷的开发向导提高 J2EE 应用开发的效率。

本章讲述如何利用 JBoss Eclipse IDE 开发 Web 版本的学生成绩管理系统，包括 JBoss Eclipse IDE 插件的安装和配置、JSP 文件的开发实例、Servlet 的开发实例，以及将开发的 Web 应用发布到 JBoss 应用服务器上的具体步骤。

6.1　下载并安装 JBoss 插件

Eclipse 提供一种简单的插件安装方法——安装/更新机制。经过第一次手工安装插件后，Eclipse 可以随时检查该插件是否有新版本，如果存在就会下载并安装新版本的插件。本节介绍如何通过 Eclipse 的安装/更新机制安装 JBoss-IDE，并自动更新插件。

跟我做

（1）启动 Eclipse。

（2）选择【帮助】|【软件更新】|【查找并安装】命令，打开【安装/更新】对话框。

（3）选中【搜索要安装的新功能部件】单选按钮，如图 6-1 所示。

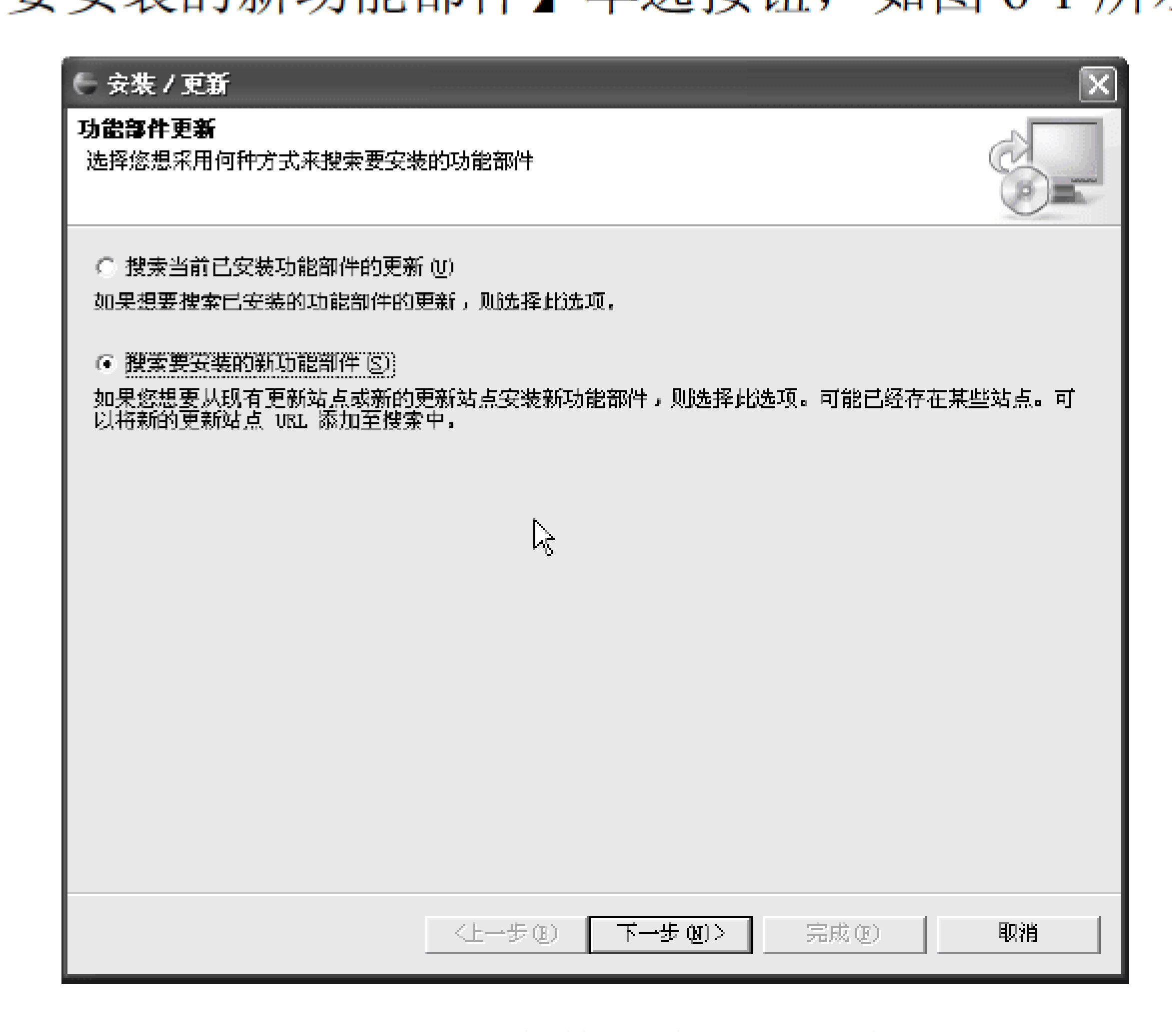


图 6-1　【安装/更新】对话框

（4）单击【下一步】按钮打开【安装】对话框，单击对话框右侧的【新建远程站点】按钮，打开【新建更新站点】对话框。

（5）将站点的名称设置为 JBoss Eclipse IDE，URL 设置为 http://download.jboss.org/jbosside/updates/stable，如图 6-2 所示。



图 6-2　新建更新站点

（6）单击【确定】按钮返回【安装】对话框。选中刚才建立的 JBoss EclipseIDE 站点，单击【完成】按钮。

（7）展开 JBoss EclipseIDE 树，选中【JBossIDE 1.6.0 GA】复选框，如图 6-3 所示。

（8）单击【下一步】按钮，Eclipse 会询问是否同意 JBoss EclipseIDE 的许可条例。如果同意，选择【我接受许可协议中的条款】。



图 6-3　【更新】对话框

（9）单击【下一步】按钮，Eclipse 将列出所有将要安装的特征（Feature），如图 6-4 所示。



图 6-4　将要安装的特征

（10）单击【完成】按钮开始安装。

（11）在所有的特征下载完毕后，Eclipse 会提示是否想要安装每个特征。这里选择【全部安装】选项。

（12）在安装完成后，会提示是否重新启动 Eclipse。单击【是】按钮重新启动，安装完毕。

6.2　配置并运行 JBoss 应用服务器

JBoss 是纯 Java 的 Web 应用服务器，为了保证 JBoss 服务器的正常运行，在安装 JBoss 之前首先要确保系统已经安装了 JDK。可以从 http://labs.jboss.com/portal/jbossas/download/index.html 下载 JBoss 应用服务器，本章选用 JBoss 4.0.4 版本。

跟我做

（1）将下载的压缩包解压至本地磁盘，例如 C:\jboss 4.0.4。解压后的 JBoss 目录结构如图 6-5 所示。

说明： bin 目录主要包含 run.jar、shutdowm.jar 等文件，用于启动、停止服务器脚本；client 目录主要包含与客户端相关的文件；docs 目录主要包含 JBoss 服务器的文档；server 目录主要包含与服务器有关的配置文件。

（2）运行 bin 目录下的 run.bat 文件，如果 DOS 界面出现类似如图 6-6 所示的信息就说明 JBoss 服务器已经启动。



图 6-5　JBoss 目录结构　　　　　　　图 6-6　JBoss 成功启动信息

（3）单击工具栏中的【调试】按钮，选择【调试】命令，打开【调试】对话框。

（4）在【调试】对话框中选择 JBoss 4.0.x（本章采用的是 JBoss 4.0.4），单击【新建】按钮。对话框右侧出现配置信息交互界面。

（5）在【名称】文本框中输入“JBoss 4.0.4”，在【JBoss 4.0.x Home Directory】文本框中选择 JBoss 应用服务器的根目录为 C:\jboss-4.0.4，【Server Configuration】选择“default”，如图 6-7 所示。



图 6-7　【调试】对话框

（6）单击【调试】按钮，JBoss 应用服务器开始启动，JBoss 服务器启动信息如图 6-8 所示。JBoss 应用服务器安装完毕。

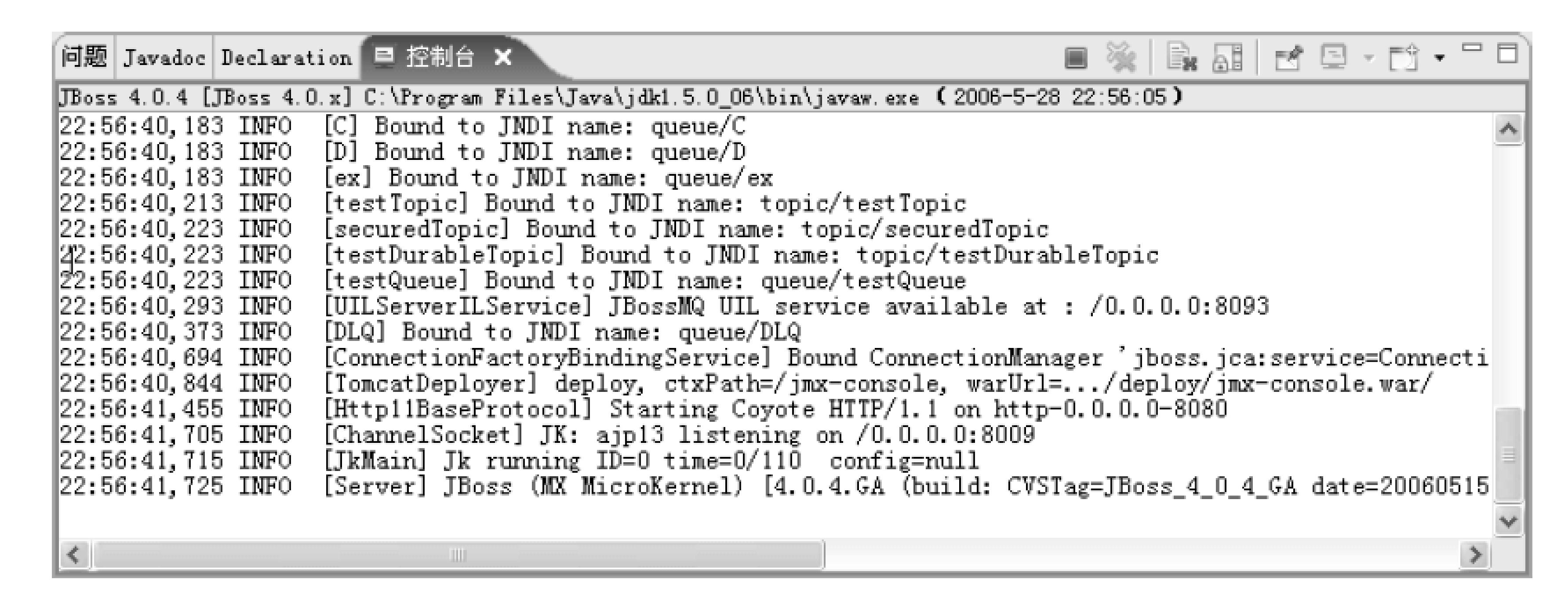


图 6-8　JBoss 服务器启动信息

6.3　在 Eclipse 中开发 Jsp

JBoss 应用服务器配置完成后，为了开发学生成绩管理 Web 应用系统，在图 6-9 中，选择 J2EE 1.4 Project，新建一个名字为 ScoreWeb 的工程。

另外，在第 5 章中介绍了 HSQLDB 的内存数据库运行模式，本章 Web 应用系统采用 HSQLDB 的独立数据库模式。

6.3.1　Eclipse 内置 JSP 编辑器的使用

Eclipse 内置了 JSP 编辑器，功能强大的 JSP 编辑器可以降低 JSP 开发的难度，增加页面开发的速度。JSP 编辑器的属性窗口可以修改 HTML 标签的相应属性，通过大纲视图可以方便地添加子标签和兄弟标签，同时编辑器具有强大的代码辅助功能。下面讲解常用的方法。

跟我做

（1）利用 Eclipse 的属性视图，如图 6-10 所示，将鼠标放在每个 HTML 标签上时，属性窗口中就会出现该标签的所有属性。



图 6-9　新建 J2EE 1.4 工程



图 6-10　属性视图

提示：可以在属性视图中直接编辑相应属性的值。

（2）在大纲视图中方便地添加每个 HTML 标签的属性和子标签、兄弟标签。

（3）JSP 编辑器提供强大的代码辅助功能，如图 6-11 所示。

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>学生成绩管理系统Web版</title>
</head>
<body>
<
 <>a          A link if the href attribute is present, and the
 <>abbr        target for a link if the name attribute is present
 <>acronym
 <>address
 <>applet
 <>b
 <>basefont
 <>bdo
```

图 6-11　代码辅助功能

6.3.2　启动数据库和创建表格

通过命令行

```
java org.hsqldb.Server –database.0 file:Scoredb –dbname.0 scoredb
```

可以启动 hsqldb 数据库。通过 hsqldb 的数据库管理器，可以执行 SQL 语句，完成创建表格、查询等数据库常用操作。

跟我做

（1）在环境变量中加入 hsqldb.jar，将%JAVA_HOME%/bin 路径设置在 Path 环境变量中。

（2）打开 CMD 窗口，输入如下命令：

```
java org.hsqldb.Server –database.0 file:Scoredb –dbname.0 scoredb
```

出现如图 6-12 所示信息，表示数据库已经成功启动。

```
D:\>java org.hsqldb.Server -database.0 file:Scoredb -dbname.0 scoredb
[Server@1270b73]: [Thread[main,5,main]]: checkRunning(false) entered
[Server@1270b73]: [Thread[main,5,main]]: checkRunning(false) exited
[Server@1270b73]: Startup sequence initiated from main() method
[Server@1270b73]: Loaded properties from [D:\server.properties]
[Server@1270b73]: Initiating startup sequence...
[Server@1270b73]: Server socket opened successfully in 10 ms.
[Server@1270b73]: Database [index=0, id=0, db=file:Scoredb, alias=scoredb] opene
d sucessfully in 301 ms.
[Server@1270b73]: Startup sequence completed in 311 ms.
[Server@1270b73]: 2006-05-29 00:55:21.904 HSQLDB server 1.8.0 is online
[Server@1270b73]: To close normally, connect and execute SHUTDOWN SQL
[Server@1270b73]: From command line, use [Ctrl]+[C] to abort abruptly
```

图 6-12　HSQLDB 数据库启动画面

这样就创建了一个数据为 Scoredb，别名为 scoredb 的数据库。

（3）通过 HSQLDB 自带的数据库管理工具管理数据库。在另一个 CMD 窗口输入如下命名：

```
java org.hsqldb.util.DatabaseManager
```

在如图 6-13 所示的窗口中输入如下连接信息。

（4）单击【OK】按钮，如果连接成功，出现如图 6-14 所示的 HSQL 数据库管理窗口。

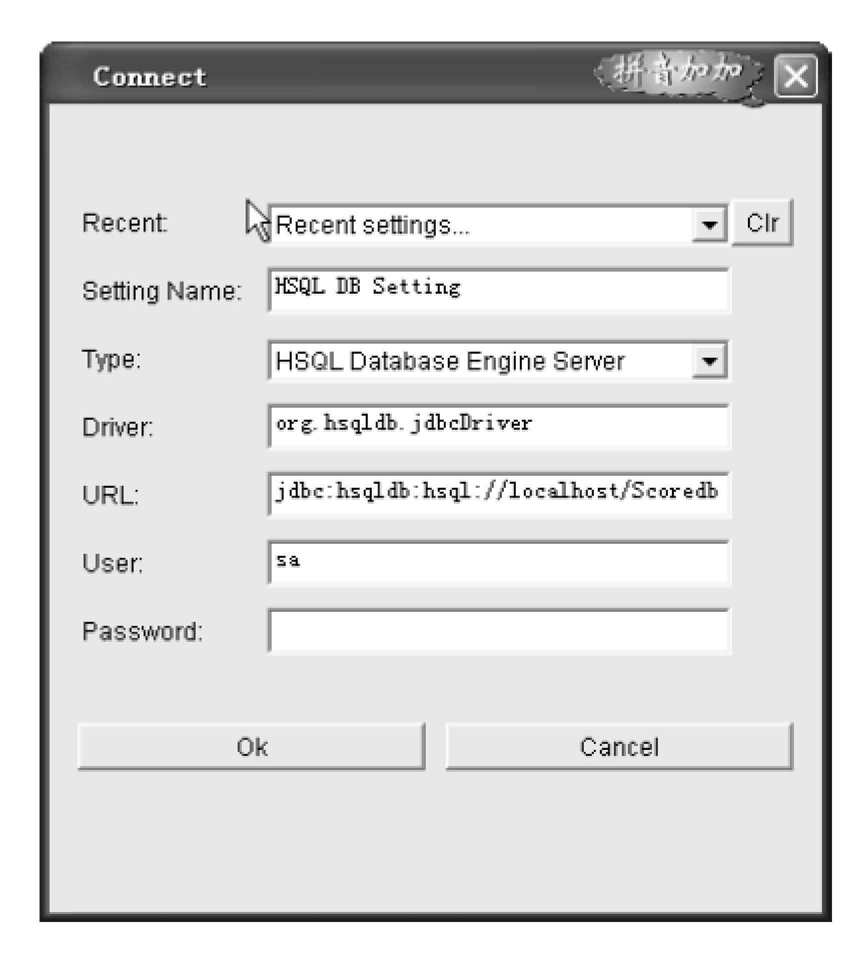


图 6-13　HSQLDB 连接信息

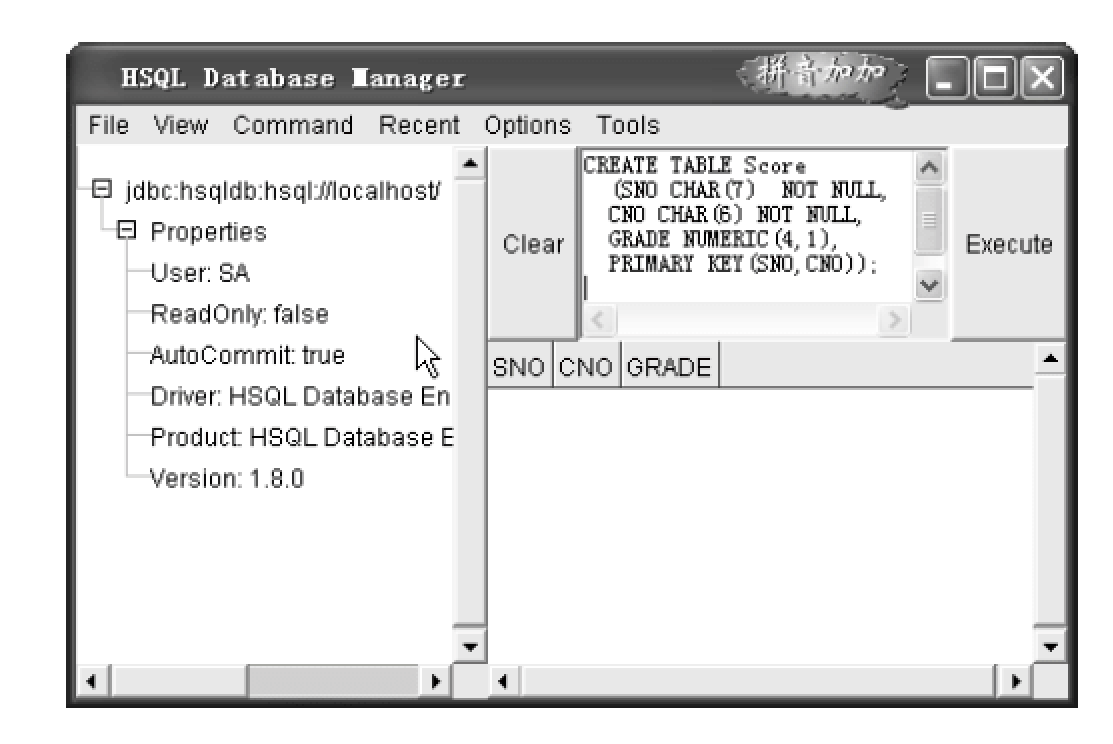


图 6-14　HSQL 数据库管理器

（5）在 SQL 语句编辑器中输入如下 SQL 语句：

```
CREATE TABLE Score
  (SNO CHAR(7)   NOT NULL,
  CNO CHAR(6) NOT NULL,
  GRADE NUMERIC(4,1),
  PRIMARY KEY(SNO,CNO));
```

（6）单击【Execute】按钮，建立表格 Score。

至此为下面章节的学习做好了所有的准备工作。

6.3.3　创建 scoreForm.jsp 录入成绩

在数据提交页面中输入相关信息，如图 6-15 所示。单击【提交】按钮后，在 HSQLDB DatabaseManager 中输入如下 SQL 语句：

```
Select   *   from   score;
```

可以看到 HSQLDB 数据库中的记录内容，如图 6-16 所示。



图 6-15　数据提交表单

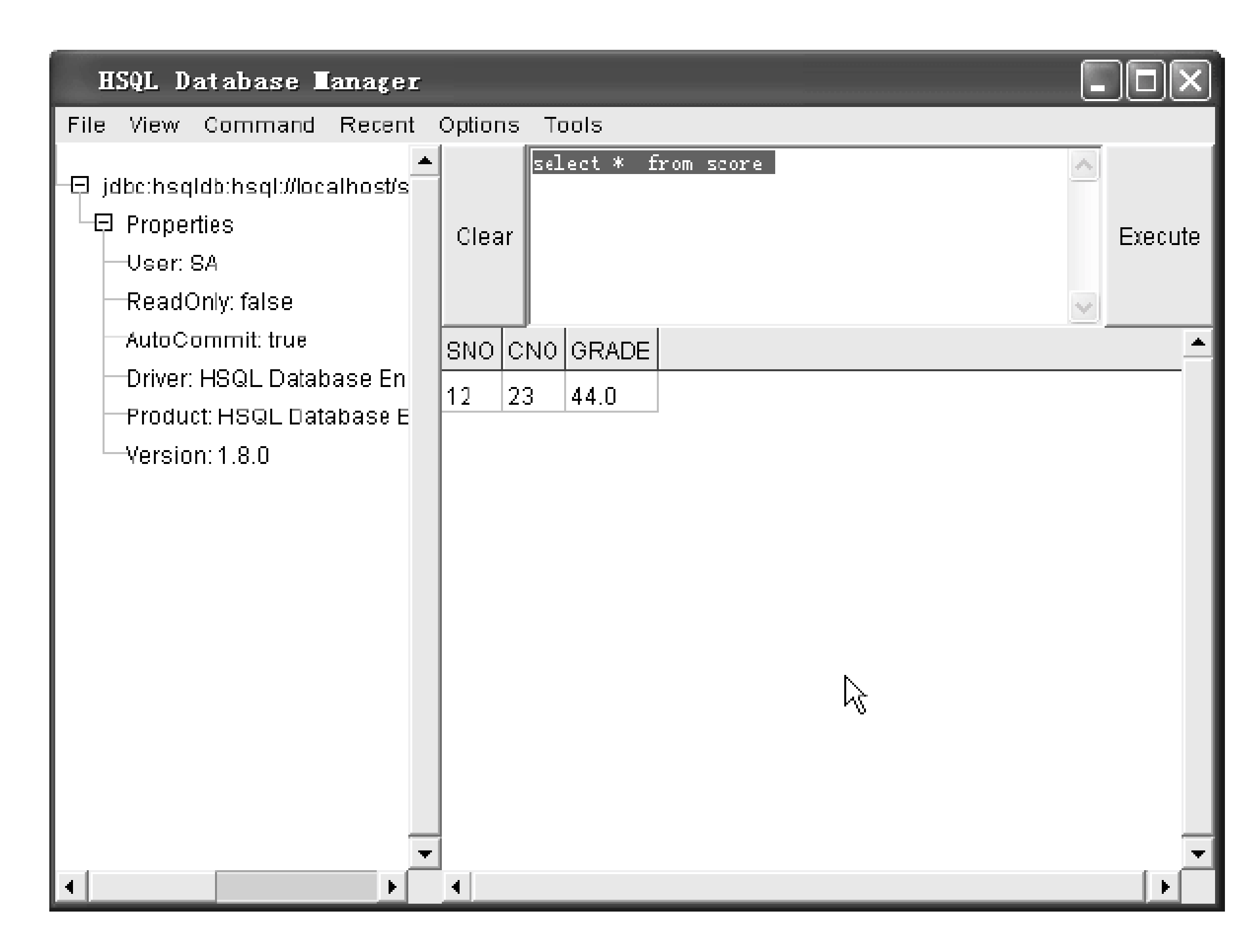


图 6-16　提交后的数据库状态

通过上面的准备，下面介绍在 JBoss Eclipse IDE 中开发 JSP 页面的具体步骤。

跟我做

（1）右击已经建立的 ScoreWeb 工程，在快捷菜单中选择【新建】|【源文件夹】命令，打开【新建源文件夹】对话框。

（2）在【文件夹名】文本框中输入“src”。单击【确定】按钮，在 ScoreWeb 工程中生成一个名字为“src”的源文件夹。

（3）右击“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（4）在【向导】树形结构中选择【其他】|【JSP】选项。单击【下一步】按钮，在【文件名】文本框中输入“scoreFrom.jsp”。

（5）单击【下一步】按钮，选择 JSP 模板“New JSP File（html）”，如图 6-17 所示。

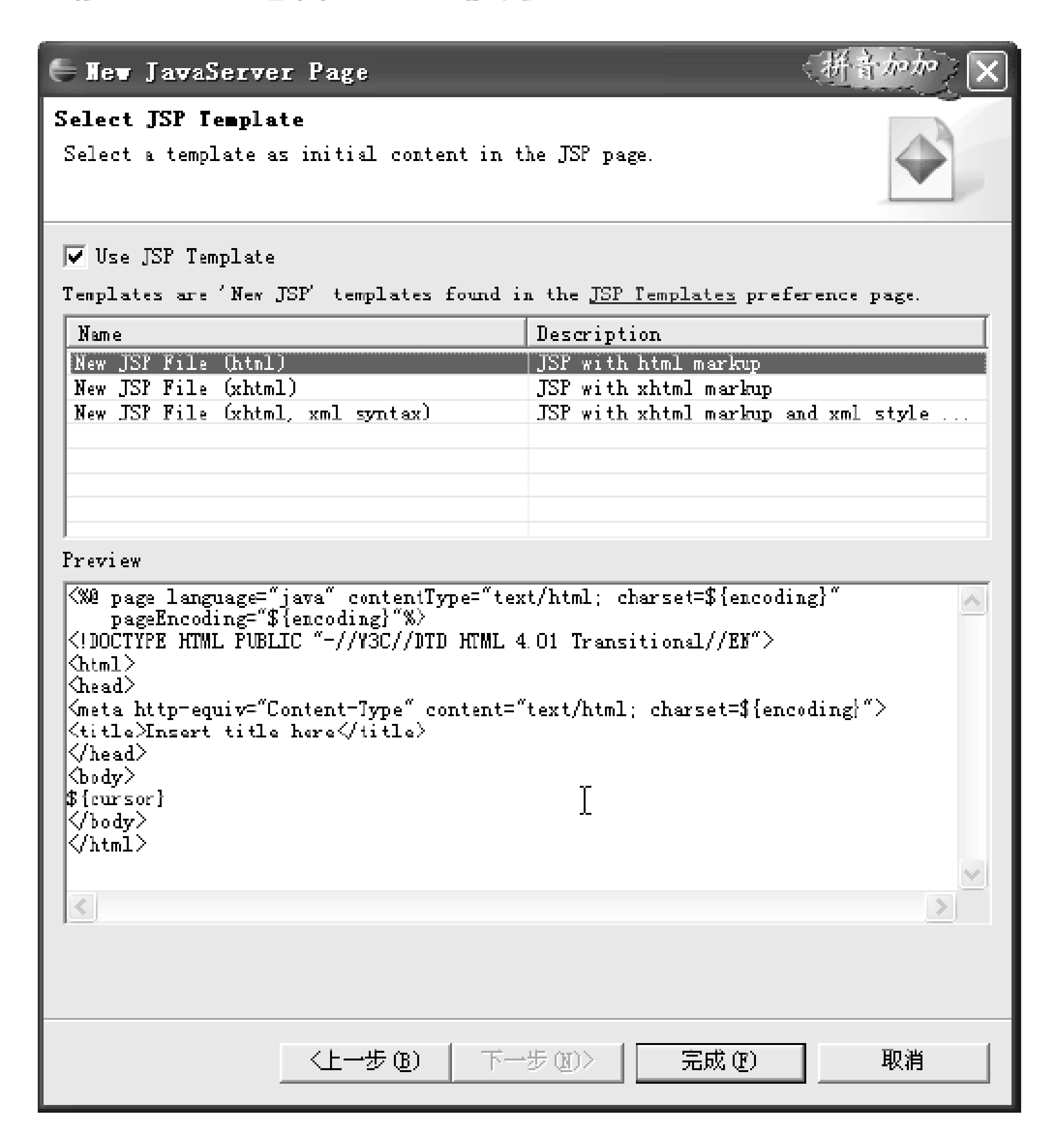


图 6-17　选择 JSP 模板

（6）单击【完成】按钮，在 ScoreWeb 工程中生成 scoreForm.jsp 文件。

（7）打开 scoreForm.jsp 文件，完成如下 JSP 代码：

```
<%@pagelanguage="java"contentType="text/html;charset=UTF-8"pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
         <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
         <title>学生成绩管理系统 Web 版</title>
    </head>
    <body>
         <!—以 POST 的方式提交学号、课号、成绩信息-->
         <form action="addScore.jsp" name="addScore" method="POST">
             <p>学号：<input type="text" name="SNO" value="0"></p>
             <p>课号：<input type="text" name="CNO" value="0"></p>
             <p>成绩：<input type="text" name="Score" value="0"></p>
             <p align="left"><input name="submit" type="submit" value="提交"></p>
         </form>
     </body>
</html>
```

由“%@...%”包含的信息是设定与 JSP 页面有关的信息，比如设定页面所使用的字符集是“UTF-8”。通过 form 的 action 属性指定表单提交后重定向到“addScore.jsp”页面。

（8）addScore.jsp 将收到的请求信息提交到 HSQLDB 数据库中。其 JSP 代码如下所示：

```
<%@pagelanguage="java"contentType="text/html;charset=ISO-8859-1"pageEncoding="ISO-8859
-1"%>
<%@page import="java.sql.*"%>
<%@page import="java.sql.*"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
    <%String SNO = "", CNO = "";
         float score;
        //取得 form 提交的 SNO、CNO 和 Score 参数的值
         SNO = request.getParameter("SNO");
         CNO = request.getParameter("CNO");
         score = Float.parseFloat(request.getParameter("Score"));
         try {
                //加载 HSQLDB 数据库 JDBC 驱动
                Class.forName("org.hsqldb.jdbcDriver");
                //连接 score 数据库，用户名为 sa，密码为空
                Connection connect = DriverManager.getConnection(
                                        "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
               //产生 Statement 对象
                Statement statement = connect.createStatement();
               //将取得的参数值写入到关系数据库中
                String sql = "insert into score values(\'" + SNO + "\'," + "\'"+ CNO + "\'," + score + ")";
```

```
              //执行 insert 语句
              statement.execute(sql);
            } catch (SQLException e) {
                e.printStackTrace();
            } catch (ClassNotFoundException e) {
                e.printStackTrace();
            }
           %>
    </head>
    <body>
    </body>
</html>
```

6.3.4　创建 scoreList.jsp 显示成绩列表

6.3.3 小节介绍了将提交的用户数据保存到数据库中，本小节介绍通过 JSP 页面查询数据库中的信息，并将查询结果显示在浏览器页面中。在浏览器中访问 scoreList.jsp 页面，查询成绩的显示效果如图 6-18 所示。



图 6-18　查询成绩

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，选择【新建】|【其他】命令，打开【新建】对话框，选择“JSP”。

（2）单击【下一步】按钮，在【文件名】文本框中输入“scoreList.jsp”。

（3）单击【下一步】按钮，选择 JSP 模板“New JSP File（html）”，如图 6-17 所示。

（4）单击【完成】按钮，scoreList.jsp 文件被建立。

（5）完成如下 JSP 代码：

```
<html>
    <title>查询结果</title>
    <body>
        <%@ page contentType="text/html;charset=gb2312"%>
        <%@ page import="java.sql.*"%>
        <%
         //装载 HSQL 数据库的驱动程序
         Class.forName("org.hsqldb.jdbcDriver");
         //根据 score 数据库的 URL、用户名和密码取得数据库的连接
         Connection con = DriverManager.getConnection(
                              "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
         //创建 Statement 对象
         Statement smt = con.createStatement();
         //查询所有记录的 SQL 语句
         String sql = "select * from score";
```

```
            //执行查询
            rs = smt.executeQuery(sql);
            //将查询结果发送到客户端
            out.println("查询结果如下:" + "<hr>");
            out.println("<table border='1'>");
            out.println("<tr bgcolor='gray'><th>学号</th><th>课程编号</th><th>成绩</th></tr>");
            while(rs.next()){
                    out.println("<tr><td>" + rs.getString(1) + "</td><td>"
                               + rs.getString(2) + "</td><td>" + rs.getFloat(3)
                               + "</td></tr>");
                    }
            con.close();
              %>
          </table>
      </body>
</html>
```

6.4　在 Eclipse 中开发 Servlet

Servlet 是运行在 Web 服务器或应用服务器上的 Java 程序，它是一个中间层，负责连接来自 HTTP 客户程序的请求和服务器上的数据库或应用程序。从某种程度上，可以将 Servlet 看作是有 HTML 的 Java 程序。本节介绍根据给定的学号查询该学生的所有课程编号以及该课程的分数的 Servlet 的实现方法和编程步骤。

6.4.1　创建 ScoreFindServlet 类查询成绩

在浏览器中输入“http://localhost:8080/scoreFind?SNO=12”，将“SNO=12”参数传递给服务器，服务器查询 HSQLDB 数据库将结果返回到客户端，其效果如图 6-19 所示。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（2）在【向导】树形结构中选择【JBoss-IDE】|【Web Components】|【HTTP Servlet】命令。

（3）单击【下一步】按钮，进入【New HTTP Servlet】对话框。

（4）在【名称】文本框中输入“ScoreFindServlet”，选中【doGet()method】复选框，如图 6-20 所示。



图 6-19　scoreFind 的运行效果

New HTTP Servlet
HTTP Servlet
不建议使用缺省包。
源文件夹(D)：ScoreWeb/src　浏览(O)...
包(K)：　(缺省)　浏览(W)...
名称(M)：ScoreFindServlet
修饰符：public　default　private　protected
abstract　final　static
超类(S)：javax.servlet.http.HttpServlet　浏览(E)...
接口(I)：　添加(A)...　除去(R)
想要创建哪些方法存根？
来自超类的构造函数(C)　继承的抽象方法(H)
init() method　service() method
destroy() method　getServletInfo() method
Which service method stubs would like to create ?
doGet() method　doPost() method
doPut() method　doDelete() method
doOptions() method　doHead() method
doTrace() method
<上一步(B)　下一步(N)>　完成(F)　取消

图 6-20　【New HTTP Servlet】对话框

（5）单击【完成】按钮，生成“ScoreFindServlet.java”。

（6）在生成的 doGet 函数体中输入如下代码：

```
//取得请求 URL 中 SNO 参数的值
String SNO = req.getParameter("SNO");
//加载 HSQLDB 数据库 JDBC 驱动程序
 Class.forName("org.hsqldb.jdbcDriver");
//建立到数据库的连接
 Connection    con = DriverManager.getConnection(
                    "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
//新建 Statement 对象
 Statement smt = con.createStatement();
 ResultSet rs =null;
  //查询 SQL 语句
 String sql = "select * from score where SNO=\"" + SNO + "\"";
  //执行数据库查询 SQL 语句
 rs = smt.executeQuery(sql);
 resp.setContentType("text/html");
 PrintWriter out = resp.getWriter();
 out.println("*****RESULT*****" + "<hr>");
 out.println("<table border='1'>");
 out.println("<tr bgcolor='gray'><th>SNO</th><th>CNO</th><th>GRADE</th></tr>");
      while (rs.next()) {
           out.println("<tr><td>" + rs.getString(1) + "</td><td>"
                        + rs.getString(2) + "</td><td>" + rs.getFloat(3)
                        + "</td></tr>");
      }
con.close();
```

从请求 URL 中取得 SNO 参数的值，根据取得的 SNO 参数值从数据库中查询符合条件的所有记录并以表格的形式在客户端显示。

6.4.2　创建 ScoreDeleteServlet 类删除成绩

本小节创建一个 Servlet，将某个学号的所有成绩都删除。

（1）在浏览器的地址栏中输入“http://localhost:8080/ScoreDelete?SNO=13”，如果数据库中不存在 SNO=13 的记录就会出现如图 6-21 所示的提示。

（2）在浏览器的地址栏中输入“http://localhost:8080/ScoreDelete?SNO=12”，如果数据库中存在 SNO=12 的记录就会出现如图 6-22 所示的返回状态。

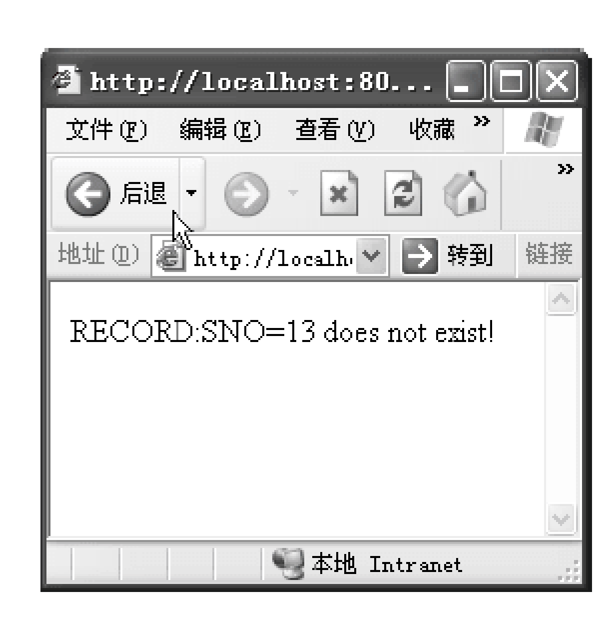


图 6-21　记录不存在的返回状态



图 6-22　删除记录

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框。

（2）在【向导】树形结构中选择【JBoss-IDE】|【Web Components】|【HTTP Servlet】命令。

（3）单击【下一步】按钮，进入下一级对话框。在【名称】文本框中输入“ScoreDeleteServlet”，这里只选中【doGet(method) 】。

（4）单击【完成】按钮，生成“ScoreDeleteServlet.java”。

（5）在生成的 doGet 函数体中输入如下代码：

```
//从请求中取得 SNO 参数的值
String SNO = req.getParameter("SNO");
//装载 HSQLDB 数据库的驱动程序
Class.forName("org.hsqldb.jdbcDriver");
//建立到数据库的连接
Connection con = DriverManager.getConnection(
                        "jdbc:hsqldb:hsql://localhost/Scoredb", "sa", "");
  //生成 Statement 对象
   Statement smt = con.createStatement();
  ResultSet rs = null;
  //查询 SNO 字段等于参数值的 SQL 语句
   String sql = "select * from score where SNO=\'" + SNO + "\'";
```

```
//执行 SQL 查询语句
 rs = smt.executeQuery(sql);
//设定响应的内容类型
 resp.setContentType("text/html");
PrintWriter out = resp.getWriter();
 //判断数据库中是否存在满足删除条件的记录
if (!rs.next()) {
       out.println("RECORD:SNO=" + SNO + " does not exist!");
  } else {
       out.println("*****Following Records Deleted*****" + "<hr>");
       out.println("<table border='1'>");
       out.println("<tr bgcolor='gray'><th>SNO</th><th>CNO</th><th>GRADE</th></tr>");
 //将即将删除的记录发送到客户端
       do {
              out.println("<tr><td>" + rs.getString(1) + "</td><td>"
                                   + rs.getString(2) + "</td><td>" + rs.getFloat(3)
                                   + "</td></tr>");
             } while (rs.next());
 //删除 SNO 字段值等于给定值的记录
       sql = "delete from score where SNO=\"" + SNO + "\"";
 //删除记录
       smt.execute(sql);
}
 //关闭连接
con.close();
```

从 Request 请求中取得参数 SNO 的值，根据取得的 SNO 值从数据库中取得所有满足条件的记录。判断返回的结果集中记录的数目。如果数目为 0，表示要删除的记录不存在，否则显示所有被删除的记录。

6.5　在 Eclipse 中开发 Filter

Filter（过滤器）是 Servlet 2.3 规范中引入的新组件类型，可以截取发送至 Servlet、JSP 页面或静态页面的请求，也可以在响应发送至客户之前先行截获。这样就可以很容易地将应用于所有请求的任务或应用所提供的服务集中起来。过滤器可以全权访问请求及响应的体和首部，因此可以完成各种不同的转换。本节假定要限制网络地址处于“10.105.20”段的用户访问上一节实现的 ScoreFindServlet，就可以为该应用添加一个 Filter。

运行效果

在浏览器的地址栏中输入“http://10.105.20.117:8080/ScoreWeb/ScoreFind?SNO=1”，因为该 URL 处于“10.105.20”网段上，所以出现如图 6-23 所示的信息。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【其他】

命令，打开【新建】对话框。

（2）在【向导】树形结构中选择【JBoss-IDE】|【Web Components】|【Filter】命令。

（3）单击【下一步】按钮，进入【New Web Filter】对话框。

（4）在【名称】文本框中输入“ScoreFilter”，其他保持默认状态，如图 6-24 所示。



图 6-23　过滤器显示的信息



图 6-24　新建过滤器

（5）单击【完成】按钮，生成 ScoreFilter 类，该类实现了 Filter 接口。

（6）在生成的 doFilter 函数中完成如下代码：

```
//取得客户端的 IP 地址
String remoteIP=request.getRemoteAddr();
 int index=remoteIP.lastIndexOf(".");
//取得客户端所处的网段
 String subIP=remoteIP.substring(0,index);
//判断是否在 10.105.20 网段上
 if(subIP.equals("10.105.20")){
        PrintWriter out=response.getWriter();
        out.println("<html><head></head><body>");
        out.println("<h1>Sorry,you can not access our application!</h1>");
        out.println("</body></html>");
        out.flush();
        return;
 }
//将控制权交给下一个 Filter
 chain.doFilter(request,response);
```

注意： 每一个 Filter 从 doFilter()方法中得到当前的 request 及 response。在这个方法中，可以进行任何的针对 request 及 response 的操作(包括收集数据,包装数据等)。Filter 调用 chain.doFilter()方法把控制权交给下一个 Filter。一个 Filter 在 doFilter()方法中结束。如果一个 Filter 想停止 request 处理而获得对 response 的完全控制，则它可以不调用下一个 Filter。

ScoreFilter 根据取得的客户端 IP 地址，判断客户端所处的网段，如果其在“10.105.20”网段上就会向客户端发送提示信息。

6.6　在 Eclipse 中开发 Listener

Listener 是 Servlet 的监听器，它可以监听客户端的请求、服务端的操作。通过监听器，可以自动激发一些操作，比如监听在线用户的数量。本节将实现一个会话事件 Listener，可以跟踪一个应用活动 Session 的个数。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【类】命令，打开【新建 Java 类】对话框。

（2）在【名称】文本框中输入“ScoreSessionListener”，作为该 Listener 类的名字。

（3）单击【添加】按钮，打开【选择已实现的接口】对话框。

（4）在【选择接口】文本框中输入“HttpSessionListener”，使得该类实现“HttpSessionListener”接口。

（5）单击【确定】按钮，返回【新建 Java 类】对话框。

（6）其他保持默认状态，如图 6-25 所示。单击【完成】按钮，生成 ScoreSessionListener 类。



图 6-25　新建 Listener 类

（7）在生成的 ScoreSessionListener 类中完成如下代码：

```
public class ScoreSessionListener implements HttpSessionListener {
	//会话个数属性的名字
	private static final String SESSION_COUNTER = "session_counter";
	//将计数器递增，该计数器作为一个 Servlet 上下文属性得到维护
	public void sessionCreated(HttpSessionEvent se) {
		int[] num = getNumber(se);
		num[0]++;
	}
	private int[] getNumber(HttpSessionEvent se) {
		HttpSession session = se.getSession();
		//取得 Session 的上下文对象
		ServletContext context = session.getServletContext();
		//取得上下文属性，名称为“session_counter”
		int[] num = (int[]) context.getAttribute(SESSION_COUNTER);
		if (num == null) {
			num = new int[1];
		//将 session_counter 属性保存到 Session 的上下文中
			context.setAttribute(SESSION_COUNTER, num);
		}
		return num;
	}
  //将计算器递减
	public void sessionDestroyed(HttpSessionEvent se) {
		int[] num = getNumber(se);
		num[0]--;
	}

}
```

ScoreSessionListener 在 Servlet 的上下文属性中维护一个计数器，当新的 Session 产生时就将计数器的值加 1，反之当 Session 完成时就将计数器的值减 1，从而监听活动 Session 的个数。

6.7　配置 web.xml 文件

为了部署 Servlet 需要配置 web.xml 文件，根据这个配置文件，JBoss 才会知道如何访问 Servlet。本节介绍如何配置 web.xml 文件，为 6.9 节的部署 Web 应用做准备。

跟我做

（1）右击“ScoreWeb”工程的“src”源文件夹，在快捷菜单中选择【新建】|【文件夹】命令，打开【新建文件夹】对话框。

（2）在对话框底部的【文件夹名称】文本框中输入“WEB-INF”。

（3）单击【完成】按钮，在 ScoreWeb 工程的“src”文件夹中生成“WEB-INF”文件夹。

（4）右击“WEB-INF”文件夹，在快捷菜单中选择【新建】|【文件】命令，打开【新建文件】对话框。

（5）在对话框底部的【文件名称】文本框中输入“web.xml”。

（6）单击【完成】按钮，在“WEB-INF”文件夹中生成“web.xml”文件。

（7）编辑 web.xml 输入如下内容：

```
<!DOCTYPE web-app PUBLIC '-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN'
'http://java.sun.com/j2ee/dtds/web-app_2.2.dtd'>
<web-app>
    <servlet>
         <!-------配置名字为 ScoreFindServlet 的 Servlet------>
        <servlet-name>ScoreFindServlet</servlet-name>
         <!------指定 ScoreFindServlet 的实现类------>
        <servlet-class>ScoreFindServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ScoreFindServlet</servlet-name>
         <!------指定 ScoreFindServlet 的 URL------>
        <url-pattern>/ScoreFind</url-pattern>
    </servlet-mapping>
    <servlet>
        <!-------配置名字为 ScoreDeleteServlet 的 Servlet------>
        <servlet-name>ScoreDeleteServlet</servlet-name>
        <!------指定 ScoreDeleteServlet 的实现类------>
        <servlet-class>ScoreDeleteServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ScoreDeleteServlet</servlet-name>
         <!------指定 ScoreDeleteServlet 的 URL------>
        <url-pattern>/ScoreDelete</url-pattern>
    </servlet-mapping>
    <filter>
         <!------配置名字为 ScoreFilter 的 Filter------>
        <filter-name>ScoreFilter</filter-name>
         <!------指定 ScoreFilter 的实现类------>
        <filter-class>ScoreFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>ScoreFilter</filter-name>
         <!------指定 ScoreFilter 的 URL------>
        <url-pattern>/ScoreFind</url-pattern>
    </filter-mapping>
    <listener>
         <!------配置名字为 ScoreFilter 的 Filter------>
        <listener-class>ScoreSessionListener</listener-class>
    </listener>
</web-app>
```

6.8　WAR 文件的打包生成

Web 应用通常都是以 WAR 文件的形式部署到 JBoss 应用服务器上。通过 JBossIDE 提供的“打包配置”功能可以定义 WAR 文件中所包含的内容。配置完成后运行该“打包配置”就可以生成所需要的 WAR 文件。本节首先介绍如何产生一个“打包配置”，然后介绍如何通过“打包配置”将本章的实例打包成一个 WAR 文件。

跟我做

（1）右击“ScoreWeb”工程，在快捷菜单中选择【属性】命令，打开【ScoreWeb 的属性】窗口。

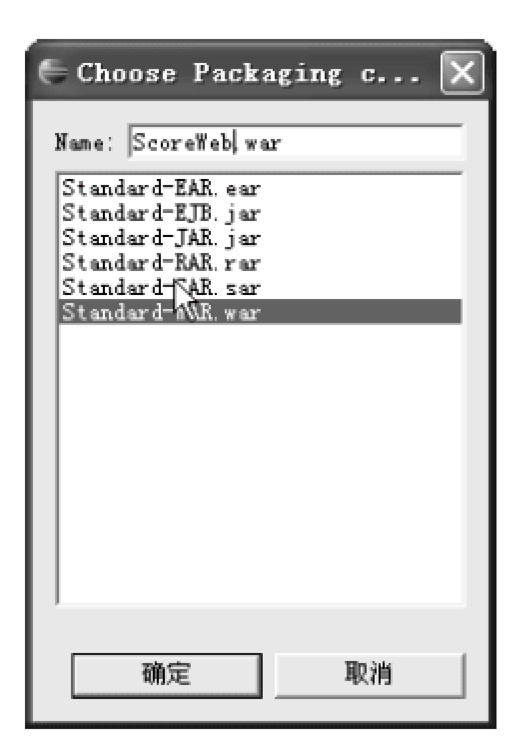
（2）在窗口左侧的树形结构中选择【Packaging Configurations】选项，窗口右侧出现“打包配置”的具体信息。

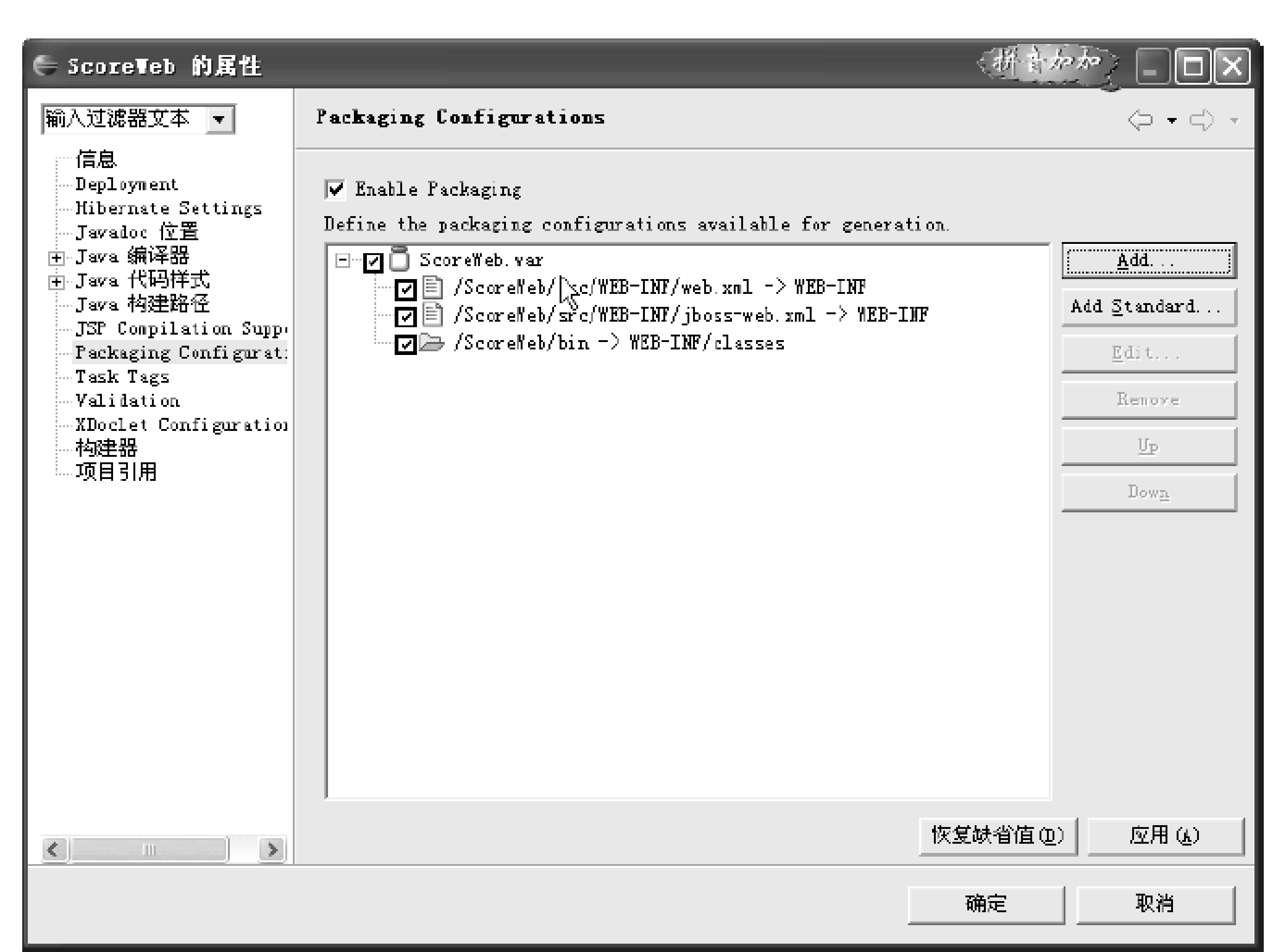
（3）单击窗口右侧的【Add Standard…】按钮，打开【Choose Packaging configurations】对话框。

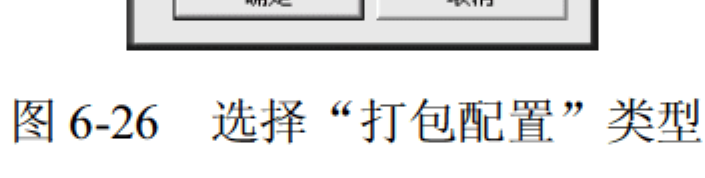
（4）选择 Standard-WAR.war，并且在【Name】文本框中输入“ScoreWeb.war”，如图 6-26 所示。

（5）单击【确定】按钮，生成名字为“ScoreWeb.war”的配置信息。

（6）展开“ScoreWeb.war”配置，选择【MANIFEST.MF】选项，单击【Remove】按钮将其删除，其最终状态如图 6-27 所示。






图 6-26　选择“打包配置”类型　　　　图 6-27　打包配置

（7）单击【确定】按钮，在“ScoreWeb”工程中生成“package-build.xml”文件。

（8）右击“ScoreWeb”工程，在快捷菜单中选择【Run Packaging】命令。该命令执行后就在“ScoreWeb”工程中生成“ScoreWeb.war”文件。

6.9　部署 Web 应用

本节将产生的 WAR 发布到 JBoss 上。

跟我做

（1）单击工具栏上的【调试】按钮，打开【调试】对话框。

（2）在对话框左侧的“配置”树形结构中选择【JBoss 4.0.x】选项，单击【新建】按钮，对话框右侧出现“JBoss 4.0.x”的配置信息。

（3）在【名称】文本框中输入“JBoss4”，作为新建“JBoss 4.0.x”的名称，单击【Browse】按钮选择 JBoss 4 的安装目录，比如“C:\jboss-4.0.4”。在【Server Configuration】下拉列表框中选择【default】选项，如图 6-28 所示。

（4）单击【应用】按钮，配置完成。

（5）右击“ScoreWeb.war”文件，在快捷菜单中选择【Deployment】|【Deploy to】命令，打开【Target Choice】对话框。

（6）选择【JBoss4】选项，如图 6-29 所示。

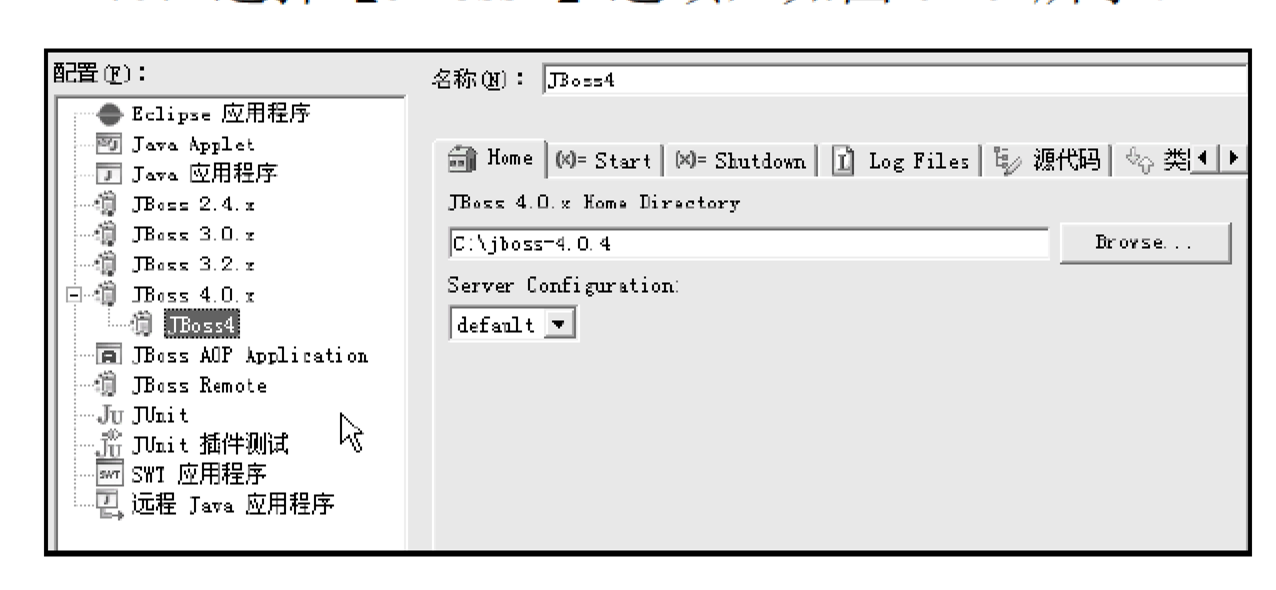



图 6-28　新建 JBoss 4.0.x 配置



图 6-29　选择目标 JBoss 服务器

（7）单击【确定】按钮，“ScoreWeb.war”文件被部署到“C:\jboss-4.0.4\server\default\deploy”目录下。

第 7 章　Struts 开发实例——在线留言板

Struts 是 Apache 软件基金会旗下 Jakarta 项目的一部分，它是一个基于 Sun J2EE 平台的 MVC 框架。Struts 把 Servlet、JSP、自定义标签和信息资源（message resources）整合到一个统一的框架中，开发人员不用再自己编码就能够实现全套 MVC 模式，提高了开发效率。而且 Struts 有着丰富的文档，所以目前 Struts 得到了普遍的应用。

本章通过“在线留言板”实例详细介绍在 Eclipse 中开发 Struts 应用的具体步骤，包括 Struts 的下载及安装、ActionForm 类的实现、Action 类的实现和 Validate 验证等内容，以期让读者可以在短时间内掌握 Struts 的主要内容，迅速入门。

7.1　下载并安装 Struts

在开发 Struts 应用之前，本节首先介绍 Struts 的下载和安装，并介绍 Struts 的目录结构、创建 Eclipse 工程并为该工程加入 Struts 的库文件。

跟我做

（1）登录 Struts 的官方站点 http://archive.apache.org/dist/struts/struts-1.2.4/，下载 Struts 框架的安装包 jakarta-struts-1.2.4.zip。

（2）将下载的安装包解压缩到设定的安装目录，如 C:\jakarta-struts-1.2.4。安装后其目录结构如图 7-1 所示。

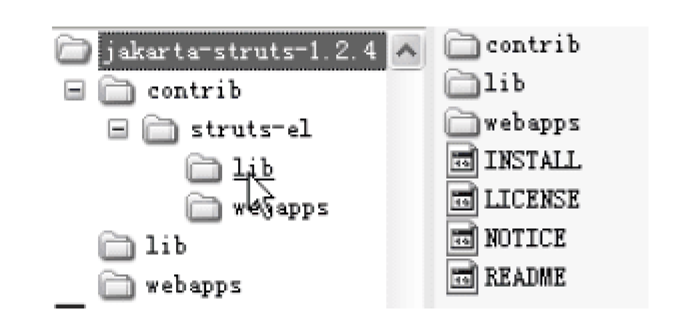


图 7-1　Struts 的目录结构

- ❑ contrib：JSTL 相关内容。
- ❑ lib：Struts 的核心 JAR 文件、Struts 依赖的第三方 JAR 文件。
- ❑ webapps：Struts 的文档和例子，是开发 Struts 应用的重要参考资料。

（3）运行 Eclipse，创建“OnlineBBS”Java 工程。在该工程中创建“WEB-INF”目录，在“WEB-INF”目录下创建“lib”目录。

（4）将 C:\jakarta-struts-1.2.4\lib 目录下的所有 JAR 文件以及%Tomcat%\common\lib 目录下的 servlet-api.jar 文件复制到“OnlineBBS”工程的“WEB-INF/lib”目录下。

（5）右击“OnlineBBS”工程，在快捷菜单中选择【属性】命令，弹出【OnlineBBS 的属性】窗口，将“WEB-INF/lib”目录下的所有 JAR 文件添加到库引用中。添加后 OnlineBBS

工程的结构如图 7-2 所示。

（6）在“OnlineBBS”工程的 WEB-INF 目录下创建“tlds”目录，将 C:\jakarta-struts-1.2.4\lib 目录下的所有 tld 文件复制到该目录下。复制后其目录结构如图 7-3 所示。

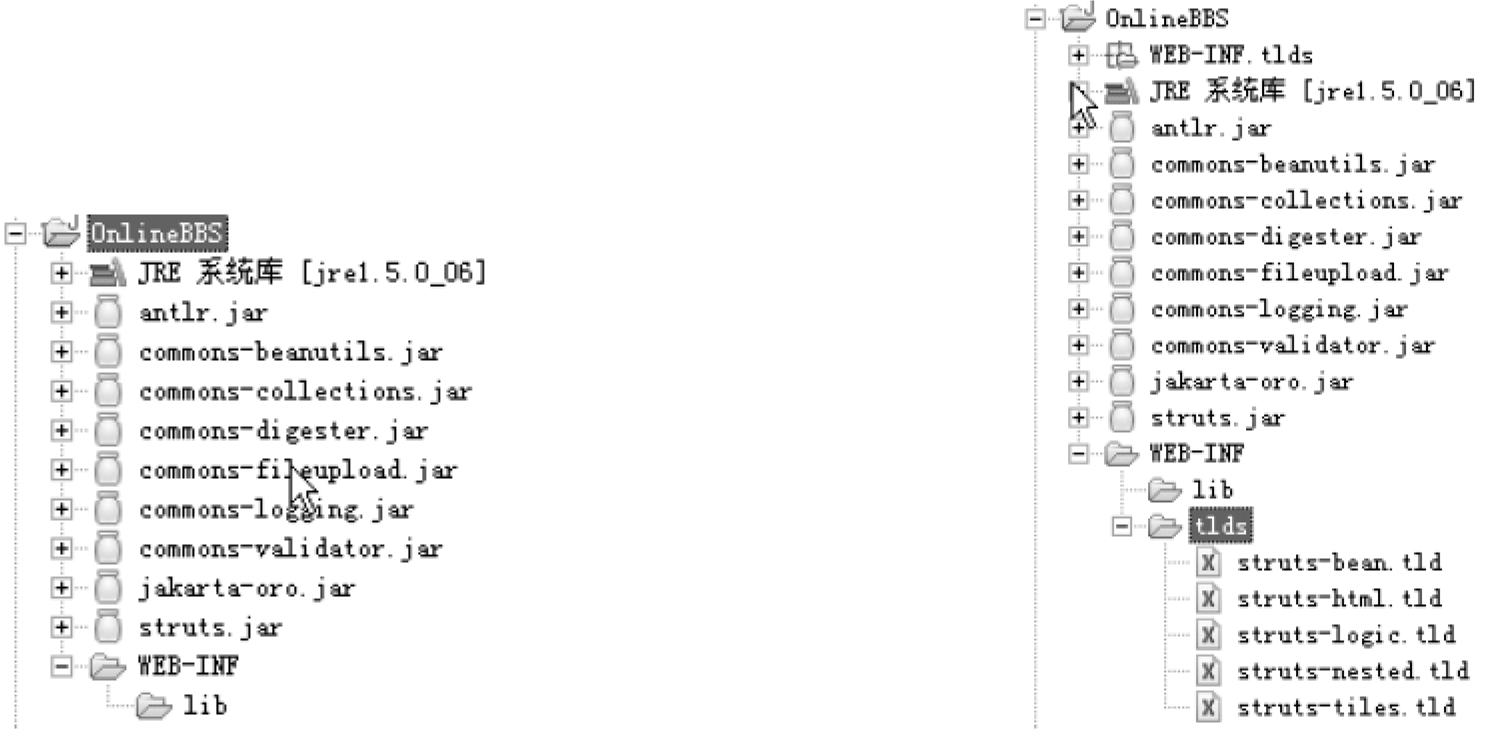


图 7-2　OnlineBBS 工程结构　　　　图 7-3　复制 tld 文件后的工程结构

（7）在“OnlineBBS”工程的 WEB-INF 目录下创建“web.xml”文件，并输入如下代码：

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
  <!--Web 应用的名称-->
  <display-name>OnlineBBS Application</display-name>
  <!--标准 action Servlet 配置  -->
  <servlet>
    <!--定义名字为 action 的 servlet-->
    <servlet-name>action</servlet-name>
    <!--servlet 对应的 Java 类名称-->
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <!--定义名字为 config 的参数-->
      <param-name>config</param-name>
      <!--参数的值指定了 Struts 核心配置文件的存放位置-->
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <!--定义名字为 debug 的参数-->
      <param-name>debug</param-name>
      <!--debug 参数的值为 2-->
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <!--定义名字为 detail 的参数-->
      <param-name>detail</param-name>
      <!--detail 参数的值为 2-->
      <param-value>2</param-value>
```

```
    </init-param>
    <load-on-startup>2</load-on-startup>
</servlet>

<!-- 标准 Action Servlet 映射-->
<servlet-mapping>
    <!--引用名字为 action 的 Servlet-->
    <servlet-name>action</servlet-name>
    <!--访问名字为 action 的 Action 的 URL 地址的后缀是*.do-->
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
<!--定义默认访问页面 -->
<welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
</welcome-file-list>
<!-- Struts 标签库定义，Tomcat 启动时，会根据这一项来加载 Struts 标签 -->
<taglib>
    <!--定义名字为/tags/struts-bean 的标签-->
    <taglib-uri>/tags/struts-bean</taglib-uri>
    <!--标签文件的存放位置-->
    <taglib-location>/WEB-INF/tlds/struts-bean.tld</taglib-location>
</taglib>

<taglib>
    <!--定义名字为/tags/struts-html 的标签-->
    <taglib-uri>/tags/struts-html</taglib-uri>
    <!--标签文件的存放位置-->
    <taglib-location>/WEB-INF/tlds/struts-html.tld</taglib-location>
</taglib>

<taglib>
    <!--定义名字为/tags/struts-logic 的标签-->
    <taglib-uri>/tags/struts-logic</taglib-uri>
    <!--标签文件的存放位置-->
    <taglib-location>/WEB-INF/tlds/struts-logic.tld</taglib-location>
</taglib>

<taglib>
    <!--定义名字为/tags/struts-nested 的标签-->
    <taglib-uri>/tags/struts-nested</taglib-uri>
    <!--标签文件的存放位置-->
    <taglib-location>/WEB-INF/tlds/struts-nested.tld</taglib-location>
</taglib>

<taglib>
    <!--定义名字为/tags/struts-tiles 的标签-->
    <taglib-uri>/tags/struts-tiles</taglib-uri>
    <!--标签文件的存放位置-->
    <taglib-location>/WEB-INF/tlds/struts-tiles.tld</taglib-location>
```

```
  </taglib>
</web-app>
```

在 URL 中出现“*.do”模式都会由 action Servlet 来负责处理，同时定义 Struts 标签。

7.2　Struts 原理简介

Struts 框架是一种典型的 MVC 实现。MVC 结构的实现原理如图 7-4 所示。本节从 MVC 结构的视图、控制器和模型 3 个方面来介绍 Struts 的基本原理。

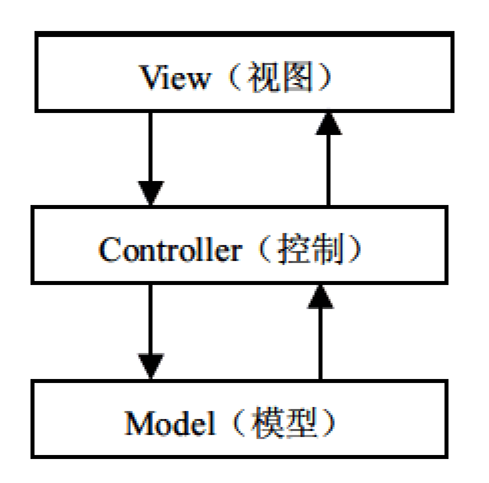


图 7-4　MVC 结构

- ❑ Struts 的视图部分：基于 Struts 框架实现的应用程序其视图部分通常采用 JSP 技术实现。JSP 页面可以包括静态 HTML 文本，也可以包含动态内容。JSP 提供了一组类似于<jsp:useBean>的标准标签，另外为方便用户快速创建用户界面，Strtus 框架包含了一系列自己定制的标签，加上用户可以自己定制的标签，所有这些组成了 Struts 框架的视图部分。
- ❑ Struts 的控制器部分：应用程序的控制器部分的主要作用是根据来自客户端（一般情况下为浏览器）的请求，判断需要执行的动作，相应动作执行完毕后将正确的视图传递给客户端。在 Struts 框架中，ActionServlet 类是关键控制器部件。所有的 Action 都继承自 org.apache.struts.action.Action 类。
- ❑ Struts 的模型部分：ActionForm 类负责封装来自 JSP 页面 Form 表单中的数据。通常，ActionForm 具有过滤保护作用，好像是 HTTP 请求和 Action 之间的防火墙，只有通过 ActionForm 验证的数据才能够发送到 Action 处理。
- ❑ Struts 的 Struts-config.xml 文件：Struts 的核心是 Controller，即 ActionServlet，而 ActionServlet 的核心是 Struts-config.xml，Struts-config.xml 集中了所有页面的导航定义。对于大型的 Web 项目，通过此配置文件即可把握其脉络。

Struts 框架的工作原理如图 7-5 所示。

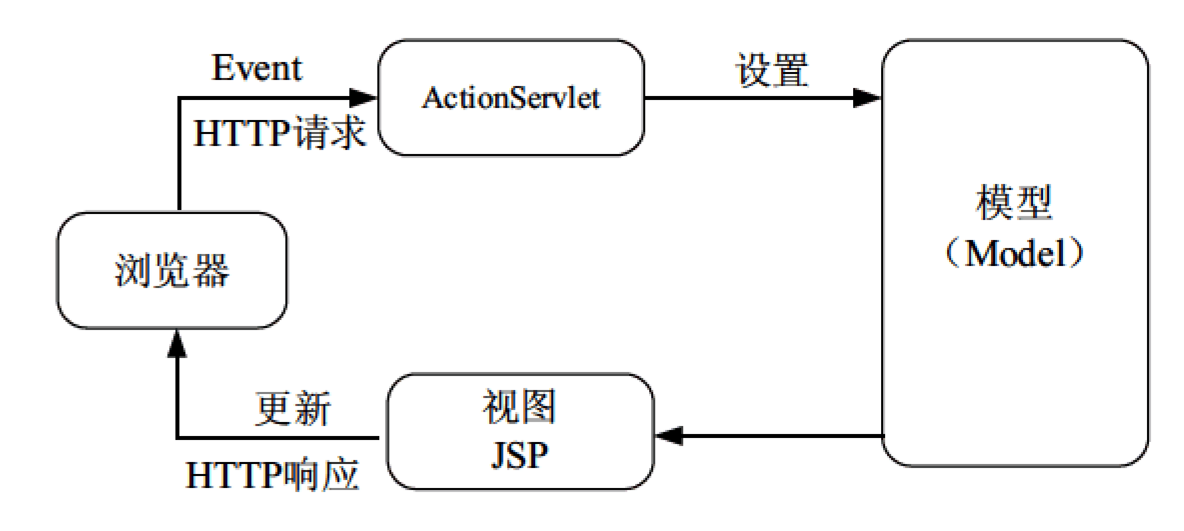


图 7-5　Servlet 的工作原理

当 Web 应用启动时加载和初始化 ActionServlet，ActionServlet 读取 struts-config.xml 文件中的配置信息。当 ActionServlet 接收到一个客户请求时，其具体执行过程步骤如下：

（1）查看用户请求的 Action，如果不存在，向用户返回路径错误信息。

（2）创建 ActionForm，将客户提交的表单数据保存到 ActionForm 对象中。

（3）如果表单需要验证，就调用 ActionForm 的 validate()方法，如果返回 null 或者返回一个不包含 ActionMessage 的 ActionErrors 对象，表示表单验证成功。

（4）ActionServlet 根据配置信息决定将请求转发给哪个 Action。如果相应的 Action 实例不存在，就先创建这个实例，然后调用 Action 的 execute()方法完成相应的动作。

（5）根据 execute()方法返回的 ActionForward 对象，ActionServlet 把客户请求转发给相应的 JSP 组件，然后返回给客户。

7.3　分析在线留言板应用的需求

在开发基于 Struts 框架的应用之前，首先从分析需求入手，枚举在线留言板应该实现的各种功能，以及限制条件。本章只是为了介绍 Strtus 框架在 Eclispe 集成开发环境中的具体开发步骤，实现了简单的在线留言板实例，其包括如下需求：

- 无用户身份认证，任何人都可以在在线留言板上张贴留言。
- 如果用户没有输入任何留言内容就提交表单，将返回出错信息，提示用户不能提交无内容的留言。
- 无用户身份认证，任何人都可以查看在线留言板上的留言列表。

7.4　使用 JSP 实现视图层

Struts 框架的视图部分以 JSP 作为实现手段，接受用户的输入并将结果反馈给用户。在线留言板实例的视图部分分为两部分：发布留言页面 messageForm.jsp 和显示留言列表页面 messageList.jsp。本节介绍这两个 JSP 页面的创建步骤。

7.4.1　创建 messageForm.jsp 发布留言

本小节介绍如何创建发布留言页面。messageFrom.jsp 提供用户界面，能够接受用户输入的留言内容。图 7-6 显示了 messageForm.jsp 提供的页面。在该页面中，当用户输入留言内容和自己的姓名后，单击【提交】按钮即可提交表单，将留言内容提交到服务器上。

跟我做

（1）在“OnlineBBS”工程中创建“pages”文件夹，在新创建的“pages”文件夹下创建“messageForm.jsp”文件。创建后其工程结构如图 7-7 所示。




图 7-6　发布留言页面

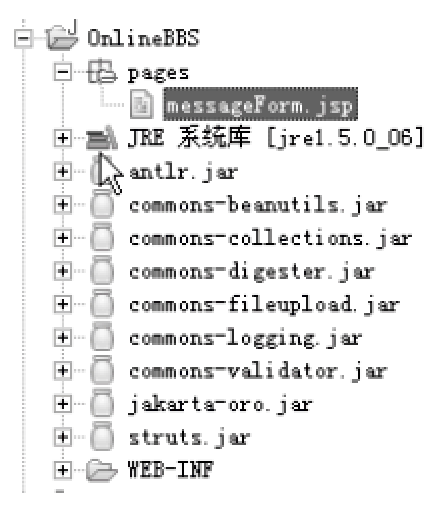


图 7-7　创建 pages 文件夹后的工程结构

（2）编辑 messageFrom.jsp 文件，在文件中输入如下代码：

```
<%@ page contentType="text/html;charset=UTF-8"language="java" %>
<!--声明和加载 Struts 标签库-->
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>

<html:html>
<head>
<!--页面的 title-->
  <title>
   <!--输出本地化的文本内容,与消息 key 匹配的文本内容来自于专门的 Resource Bundle-->
   <bean:message key="messageForm.jsp.title"/>
  </title>
<html:base/>
</head>
<body bgcolor="white">
<!--用于创建 HTML 表单，它能够将 HTML 表单的字段和 ActionForm Bean 的属性关联起来-->
<html:form action="input">

<table border="0" width="100%">
  <tr>
    <th align="right">
      <!--显示“留言内容”-->
      <bean:message key="messageForm.jsp.note"/>
    </th>
    <td align="left">
     <!--输入留言内容文本框-->
      <html:textarea property="content" cols="40" rows="8"/>
    </td>
  </tr>
  <tr>
    <th align="right">
```

```
            <!--显示“我的名字”-->
            <bean:message key="messageForm.jsp.name"/>
        </th>
        <td align="left">
            <!--创建 HTML 表单的文本框-->
            <html:text property="author" size="51" />
        </td>
    </tr>
    <tr/>
    <tr/>
    <tr>

           <td/>
           <td align="middle">
            <!--提交按钮-->
            <html:submit property="submit">
                <bean:message key="button.save"/>
            </html:submit>
             
            <!--重置按钮-->
            <html:reset>
                <bean:message key="button.reset"/>
            </html:reset>
             
            <!--取消按钮-->
            <html:cancel>
                <bean:message key="button.cancel"/>
            </html:cancel>
        </td>
    </tr>
</table>

</html:form>

</body>
</html:html>
```

本例中所有的文本内容都使用<bean:message>标签来输出。这些文本来自于 Resource Bundler，每个 Resource Bundle 都对应一个或多个本地化的消息资源文件，所以需要创建资源文件。

（3）在“OnlineBBS”工程的“WEB-INF”目录下创建“src”文件夹，在新创建的“src”文件夹中创建“MessageResources.properties”文件。创建后其目录结构如图 7-8 所示。

（4）编辑 MessageResources.properties 文件，输入如下代码：



图 7-8　创建“src”文件夹后的工程结构

```
button.cancel=取消
button.reset=重置
button.save=提交
messageForm.jsp.title=在线留言板
messageForm.jsp.name=我的名字:
messageForm.jsp.note=留言内容:
```

7.4.2　创建 messageList.jsp 显示留言列表

通过 messageForm.jsp 页面发布的留言可以通过 messageList.jsp 页面来查看，在 messageList.jsp 页面中显示所有留言者姓名和留言内容。其运行效果如图 7-9 所示。



图 7-9　留言列表

跟我做

（1）在“OnlineBBS”工程中的“pages”文件夹下创建“messageList.jsp”文件。

（2）打开 messageList.jsp 文件，输入如下代码：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!--声明和加载 Struts 标签库-->
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<html:html>
<body bgcolor="white">
<table border="1" width="100%">
<!--HTML 表格的表头，共有两列-->
  <tr>
    <!--“内容”列占页面宽度的 70%-->
    <th align="center" width="70%">
      <!--从资源文件中读取列的名字为“内容”-->
        <bean:message key="note.content"/>
    </th>
    <!--“作者”列占页面宽度的 30%-->
```

```
      <th align="center" width="30%">
      <!--从资源文件中读取列的名字为“作者”-->
        <bean:message key="note.author"/>
      </th>
    </tr>
  <!--根据 notebean 的 getNotes()方法返回的数组循环生成包含在该标记之间的 HTML 代码-->
<logic:iterate id="note" name="notebean" property="notes">
    <tr>
      <td align="left">
        <!--取得 bean 的 content 属性的值-->
        <bean:write name="note" property="content" filter="true"/>
      </td>
      <td align="left">
        <!--取得 bean 的 author 属性的值-->
        <bean:write name="note" property="author" filter="true"/>
      </td>
    </tr>
</logic:iterate>

</table>
</body>
</html:html>
```

messageList.jsp 页面只是一个简单的表格，该表格共有两列，分别表示留言的内容和作者姓名。“logic:iterate”标签会根据 notebean 的 getNotes()方法所返回的数组循环生成包含在其之间的 HTML 代码，从而可以将所有的留言以二维表格的方式展现出来。

（3）修改 MessageResources.properties 资源文件。在文件的末尾加入如下代码：

```
note.content=内容
note.author=作者
```

新加入的资源条目表示表格的两个列名。

7.5　创建 ActionForm

ActionForm 在 Struts 中位于视图组件和控制器组件之间，用于传递 HTML 表单中的数据。通常每个 HTML 表单对应一个 ActionForm，并且 ActionForm 的属性与 HTML 表单中的字段一一对应。ActionForm 可以对用户输入的数据进行合法性验证。本节将创建 MessageForm 类，用于传递用户输入的留言内容和用户姓名。

实现 MessageForm 类

该类是典型的 JavaBean，包括两个属性 content 和 author，分别对应于留言的内容和作者，为两个属性都提供了 getter/setter 方法。通过在 Struts 配置文件中进行适当的配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。本小节实现

MessageForm 类，它继承 org.apache.struts.validator.ValidatorForm 类。

跟我做

（1）右击“OnlineBBS”工程，在快捷菜单中选择【新建】|【源文件夹】命令，打开【新建源文件夹】对话框，在【文件夹名】文本框中输入“src”，单击【确定】按钮，创建名字为“src”的源文件夹。

（2）右击“src”源文件夹，在快捷菜单中选择【新建】|【类】命令，打开【新建 Java 类】对话框。在【名称】文本框中输入“MessageForm”。单击【超类】文本框后面的【浏览】按钮，打开【超类选择】对话框。

（3）选择 org.apache.struts.validator.ValidatorForm 类，单击【确定】按钮，如图 7-10 所示。单击【新建 Java 类】对话框的【完成】按钮，创建名字为 MessageForm.java 的类。



图 7-10　超类选择

（4）编辑新建的 MessageForm.java 类，输入如下代码：

```
import org.apache.struts.validator.ValidatorForm;

public class MessageForm extends ValidatorForm {

    //存放留言内容
    private String content;

    //留言者姓名
    private String author;

    /**
     * 取得作者姓名
     * @return
     */
    public String getAuthor() {
        return author;
    }

    /**
     * 设置作者姓名
     * @param author
```

```
     */
    public void setAuthor(String author) {
        this.author = author;
    }

    /**
     * 取得留言的内容
     * @return
     */
    public String getContent() {
        return content;
    }

    /**
     * 设置留言内容
     * @param content
     */
    public void setContent(String content) {
        this.content = content;
    }


}
```

7.6　使用 Action 类实现控制层

当用户在 messageForm.jsp 页面中提交留言表单后，Struts 框架就会把用户请求转发给 Action 组件。本节实现在线留言板的 Action 类，实现其控制层。

7.6.1　实现 MessageFormAction 类

MessageForm 将 messageForm.jsp 页面中用户的输入信息提交给 MessageFormAction 类来处理。本小节将实现这个 Action 类，其完成的操作是将 MessageForm 传递过来的消息保存到一个 XML 文件中，用户提交留言后将跳转到 messageList.jsp 页面列出所有的留言。

跟我做

（1）在"OnlineBBS"工程中创建 MessageFormAction 类，继承于 org.apache.struts.action.Action 类。

（2）编辑 MessageFormAction 类，输入如下代码：

```
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

/**
 * messageForm.jsp 对应的 Action，将留言保存到 xml 中
 *
 */
public class MessageFormAction extends Action {

    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws Exception {
        // 将参数 form 进行类型转换
        MessageForm messageForm = (MessageForm) form;
        // 取得 MessageForm 传递过来的留言内容
        String content = messageForm.getContent();
        // 取得 MessageForm 传递过来的留言作者
        String author = messageForm.getAuthor();
        // 将留言的内容和作者写入 note.xml 文件中

        try {
           //创建一个 File 对象
            File file = new File("note.xml");
            // 如果文件不存在就创建一个文件
            boolean success = file.createNewFile();

            // 生成文件输出流
            BufferedWriter out = new BufferedWriter(new FileWriter(file, true));
            // 将留言写到 note.xml 文件中，每条留言都是以 author=content 的形式来保存的
            out.write(author + "=" + content+"\n");
            // 关闭文件输出流
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        // 将页面跳转到 messageList.jsp 页面
        return mapping.findForward("list");
    }
}
```

MessageFormAction 将用户通过 messageForm.jsp 页面输入的内容保存到一个 note.xml 文件中，在保存文件时如果 note.xml 不存在将首先建立该文件。

7.6.2　实现 MessageListAction 类

本小节创建显示所有留言的 Action，该 Action 从 note.xml 文件中读取所有的用户留言，把这些留言的内容通过 HttpSession 传递给 messageList.jsp 页面，页面 Session 中取得留言后以二维表格的方式将它们展现出来。

跟我做

（1）右击“OnlineBBS”工程的“src”文件夹，创建 Note.java 类。

（2）打开新建的 Note.java 类，输入如下代码：

```
public class Note {

	//留言的内容属性
	private String content;
	//留言的作者
	private String author;

	/**
	 * 取得留言的作者
	 * @return
	 */
	public String getAuthor() {
		return author;
	}

	/**
	 * 设置留言的作者
	 * @param author
	 */
	public void setAuthor(String author) {
		this.author = author;
	}

	/**
	 * 取得留言的内容
	 * @return
	 */
	public String getContent() {
		return content;
	}

	/**
	 * 设置留言的内容
	 * @param content
	 */
	public void setContent(String content) {
```

```
            this.content = content;
        }

}
```

该类为标准的 JaveBean 类，包括两个字符串属性 content 和 author，分别对应于留言的内容和作者，并且为每个属性都提供了 getter/setter 方法。

（3）创建 Notes.java 类，输入如下代码：

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
public class Notes {
        /**
         * 从 note.xml 文件读取所有的留言
         * @return
         */
        public Note[] getNotes() {
                //存放所有留言的链表
                List notesList = new ArrayList();
                try {
                        //声明一个文件输入流，读取 note.xml 文件
                        BufferedReader in = new BufferedReader(new FileReader("note.xml"));
                        String str;
                        //循环处理文件的每一行
                        while ((str = in.readLine()) != null) {
                                //将文件两端的多余空格去掉
                                str = str.trim();
                                //声明一个 Note 类实例
                                Note note = new Note();
                                //将留言内容和作者分别赋值给 Note 类的相应属性
                                note.setAuthor(str.substring(0, str.indexOf("=")));
                                note.setContent(str.substring(str.indexOf("=") + 1));
                                //将留言加入到链表中
                                notesList.add(note);
                        }
                        in.close();
                } catch (IOException e) {
                }
              //将留言列表转换成数组
                Note[] result = new Note[notesList.size()];
                for (int index = 0; index < notesList.size(); index++) {
                        result[index] = (Note) notesList.get(index);
                }
               //返回一个留言的数组
                return result;
        }
}
```

该类只有一个方法 getNotes()，该方法返回 note.xml 文件中的所有留言。

（4）创建 MessageListAction.java 类，输入如下代码：

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class MessageListAction extends Action {

	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response)
			throws Exception {
		// 从 request 中取得 session
		HttpSession session = request.getSession();
		// 声明一个 Notes 类的对象
		Notes notes = new Notes();
		// 将 Notes 类的对象保存到 session 中
		session.setAttribute("notebean", notes);
		// 将页面跳转至显示留言页面
		return mapping.findForward("note");
	}

}
```

MessageListAction 将 Notes 类的实例保存到 Session 中，该实例包含了所有的留言信息，然后将页面跳转至显示所有留言页面。

7.7　生成 Struts 配置文件

Struts 的核心是控制器，即 ActionServlet。而 ActionServlet 的核心是 Struts-config.xml，Struts-config.xml 集中了所有页面的导航定义。掌握 Struts-config.xml 是掌握 Struts 的关键所在。本节创建 Struts 的配置文件，将 JSP 页面、Action 等组件结合起来。

跟我做

在“OnlineBBS”工程中创建“struts-config.xml”文件，在文件中输入如下代码：

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC
          "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
          "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">
<struts-config>
```

```
<!--FromBean 的定义-->
<form-beans>
  <!--messageFrom.jsp 对应的 FormBean-->
    <form-bean
        name="messageForm"
        type="MessageForm"/>
  <!--messageList.jsp 对应的 FormBean-->
    <form-bean
        name="listForm"
        type="org.apache.struts.validator.ValidatorForm"/>
</form-beans>
<!--定义全局页面的跳转-->
<global-forwards>
<!--定义名称为 list 的页面跳转-->
    <forward
        name="list"
        path="/messageList.do"/>
<!--定义名称为 note 的页面跳转-->
    <forward
        name="note"
        path="/listnote.do"/>
</global-forwards>
<!-- Action 映射定义 -->
<action-mappings>
    <!-- 定义 path 为"/publish"的 Action -->
    <action
        path="/publish"
        forward="/pages/messageForm.jsp"/>
    <!-- 定义 path 为"/listnote"的 Action -->
    <action
        path="/listnote"
        forward="/pages/messageList.jsp"/>
    <!-- 定义 path 为"/input"的 Action -->
   <action
        path="/input"
        type="MessageFormAction"
        name="messageForm"
        scope="request"
        validate="true"
        input="/pages/Welcome.jsp"/>
    <!-- 定义 path 为"/messageList"的 Action -->
    <action
         path="/messageList"
         type="MessageListAction"
         name="listForm"
         scope="request"
         validate="true"
         input="/pages/Welcome.jsp"/>
```

```
    </action-mappings>
<!-- Controller 配置  -->
    <controller
        processorClass="org.apache.struts.tiles.TilesRequestProcessor"/>
    <!--  资源文件的配置  -->
    <message-resources parameter="MessageResources" />
    <!-- TilesPlugin 的配置  -->
    <plug-in className="org.apache.struts.tiles.TilesPlugin" >
      <!-- XML 定义文件的路径  -->
      <set-property property="definitions-config"
                              value="/WEB-INF/tiles-defs.xml" />
      <!--  设置 Module-awareness 为 true -->
      <set-property property="moduleAware" value="true" />
    </plug-in>
  <!-- Validator plugin  的设置-->
  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property
         property="pathnames"
         value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
    </plug-in>
</struts-config>
```

该配置文件将在线留言板的各个部分组合起来，其基本流程如图 7-11 所示。MessageForm 将用户通过 messageForm.jsp 页面提交的留言内容和作者姓名等信息提交给 MessageFormAction，该 Action 将收到信息保存到 note.xml 文件中。同时将控制权转移到 MessageListAction，该 Action 从 note.xml 文件中读取所有的留言内容，将这些信息在 messageList.jsp 页面中显示出来。

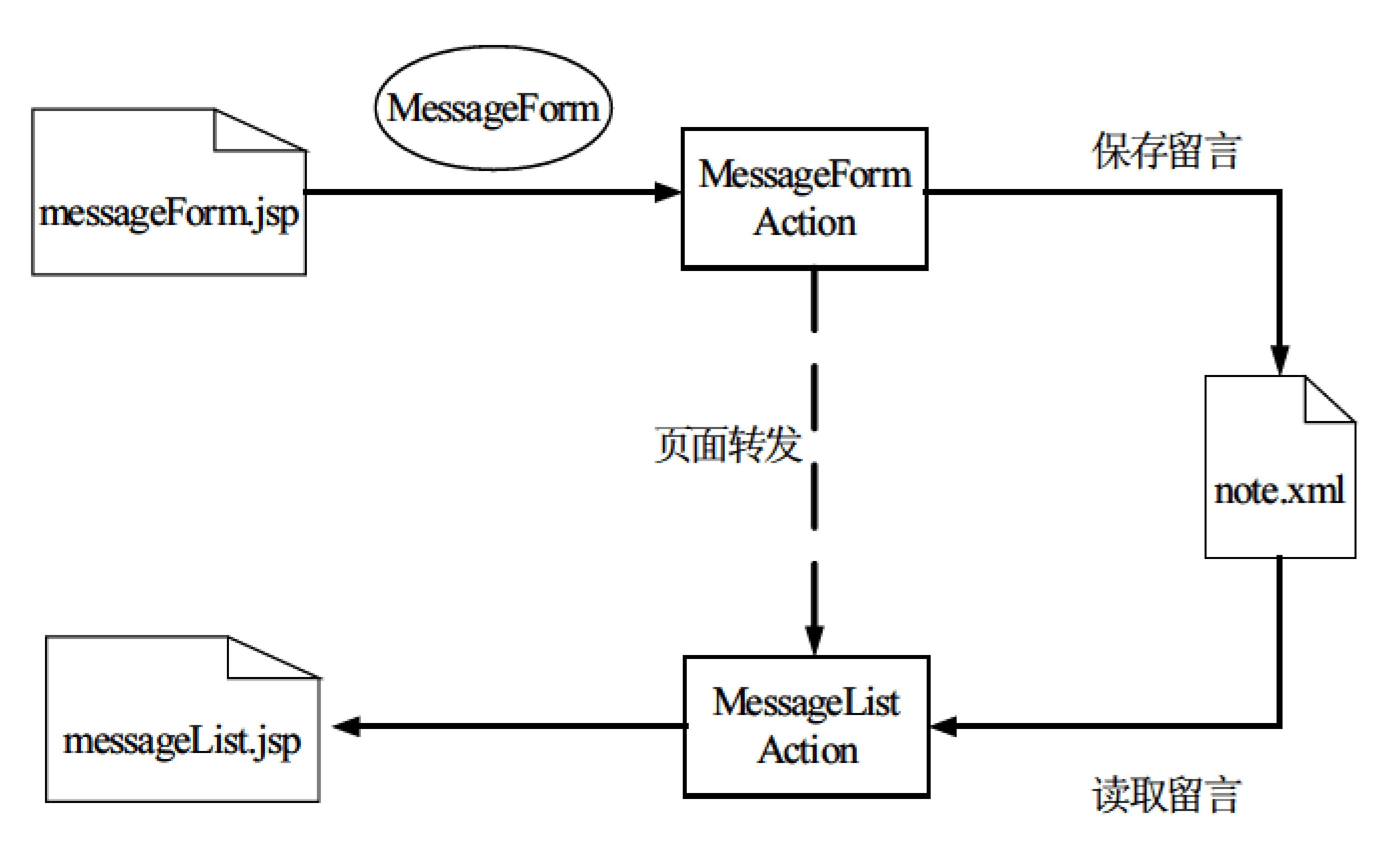


图 7-11　在线留言板模块关系图

7.8　在线留言板的 Tomcat 部署

在线留言板的部署可以通过 Ant 自动打包成 War 文件。具体方式可以参考第 4 章的说明。本节介绍一种快速的将在线留言板部署到 Tomcat 中的方式。

跟我做

（1）将 C:\jakarta-struts-1.2.4\webapps\struts-blank.war 文件复制到%Tomcat%/webapps 目录下，启动 Tomcat，该 war 文件将自动解压，在 webapps 目录下生成 struts-blank 文件夹。

（2）将 struts-blank 文件夹重新命名为 OnlineBBS，其目录结构如图 7-12 所示。

（3）将 messageForm.jsp 和 messageList.jsp 页面复制到 pages 目录下。

（4）复制 OnlineBBS 工程中的 struts-config.xml 文件覆盖 OnlineBBS\WEB-INF 目录下的 struts-config.xml 文件。

（5）将 Eclipse 切换至资源透视图，将 OnlineBBS 工程中 bin 目录下的所有 class 文件如图 7-13 所示复制到 OnlineBBS\WEB-INF\classes 目录下。

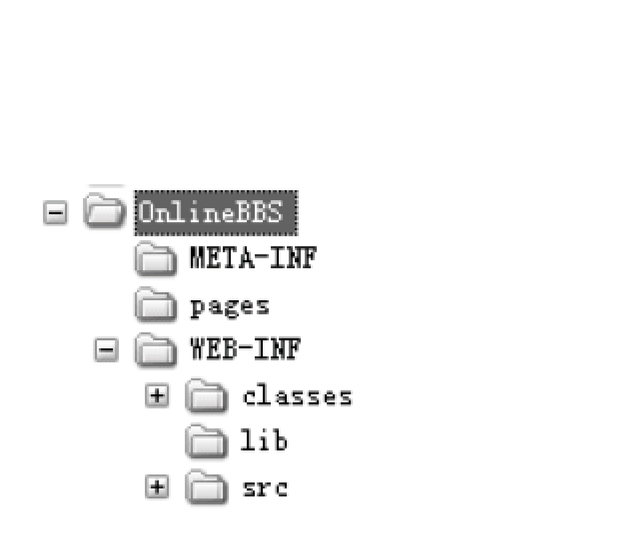


图 7-12　OnlineBBS 目录结构

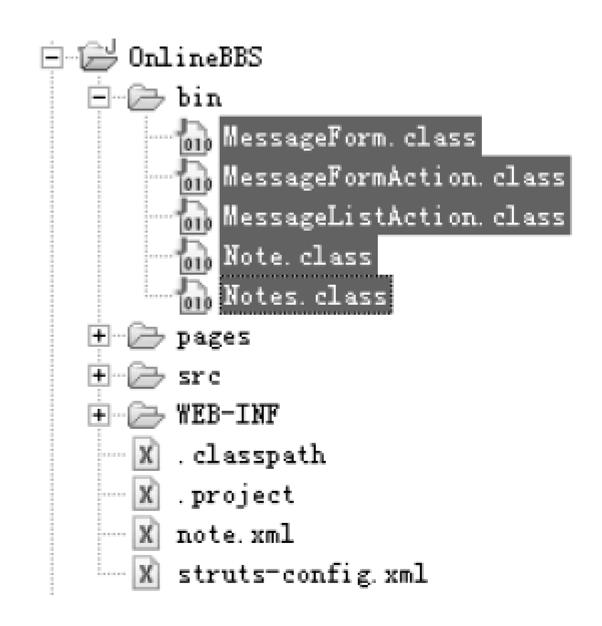


图 7-13　bin 目录下的 class 文件

（6）复制 OnlineBBS 工程的 MessageResource.properties 文件到%Tomcat%\webapps\OnlineBBS\ WEB-INF\classes 目录下，覆盖原先的同名文件。

（7）复制 OnlineBBS 工程 WEB-INF 目录下的 web.xml 文件到%Tomcat%\webapps\OnlineBBS\ WEB-INF\目录下，覆盖原先的同名文件。

（8）复制 OnlineBBS 工程 WEB-INF 目录下的 tlds 文件夹到%Tomcat%\webapps\OnlineBBS\ WEB-INF\目录下。

7.9　在浏览器中运行实例

本节在浏览器中查看在线留言板的实际运行效果。

跟我做

（1）启动 Tomcat，打开 IE 浏览器，在地址栏中输入“http://localhost:8080/OnlineBBS/publish.do”，运行界面如图 7-14 所示。

（2）在【留言内容】文本框中输入“This is note1”，在【我的名字】文本框中输入“wang”，如图 7-15 所示。



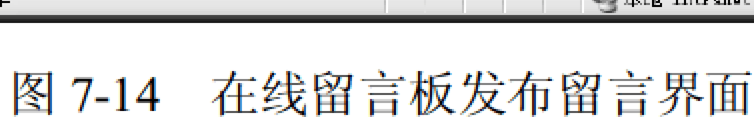

图 7-14　在线留言板发布留言界面



图 7-15　输入留言内容和作者姓名

（3）单击【提交】按钮，将所有的留言和作者姓名都展现出来，如图 7-16 所示。



内容	作者
This is note1.	Tom
This is note2.	Jack
This is note3	Mike
This is note1	wang

图 7-16　所有留言列表

（4）打开%Tomcat%/bin 目录下的 note.xml 文件，内容如下：

```
Tom=This is note1
Jack=This is note2
Mike=This is note3
wang=This is note1
```

该文件中的内容就是用户提交的留言内容。该文件相当于整个应用的 Model 部分。

7.10　使用 validator 进行留言内容验证

在所有与用户交互的应用中数据验证是必不可少的。Struts 框架提供了现成的数据验证功能。本节利用 Struts 框架的验证功能完成对数据的验证：不允许发布内容为空的留言和不允许留言者不署名。

跟我做

（1）打开“OnlineBBS”工程中的“MessageForm.java”文件，右击 Java 文件的任何

位置，在快捷菜单中选择【源代码】|【覆盖/实现方法】命令，打开【覆盖/实现方法】窗口，如图 7-17 所示。

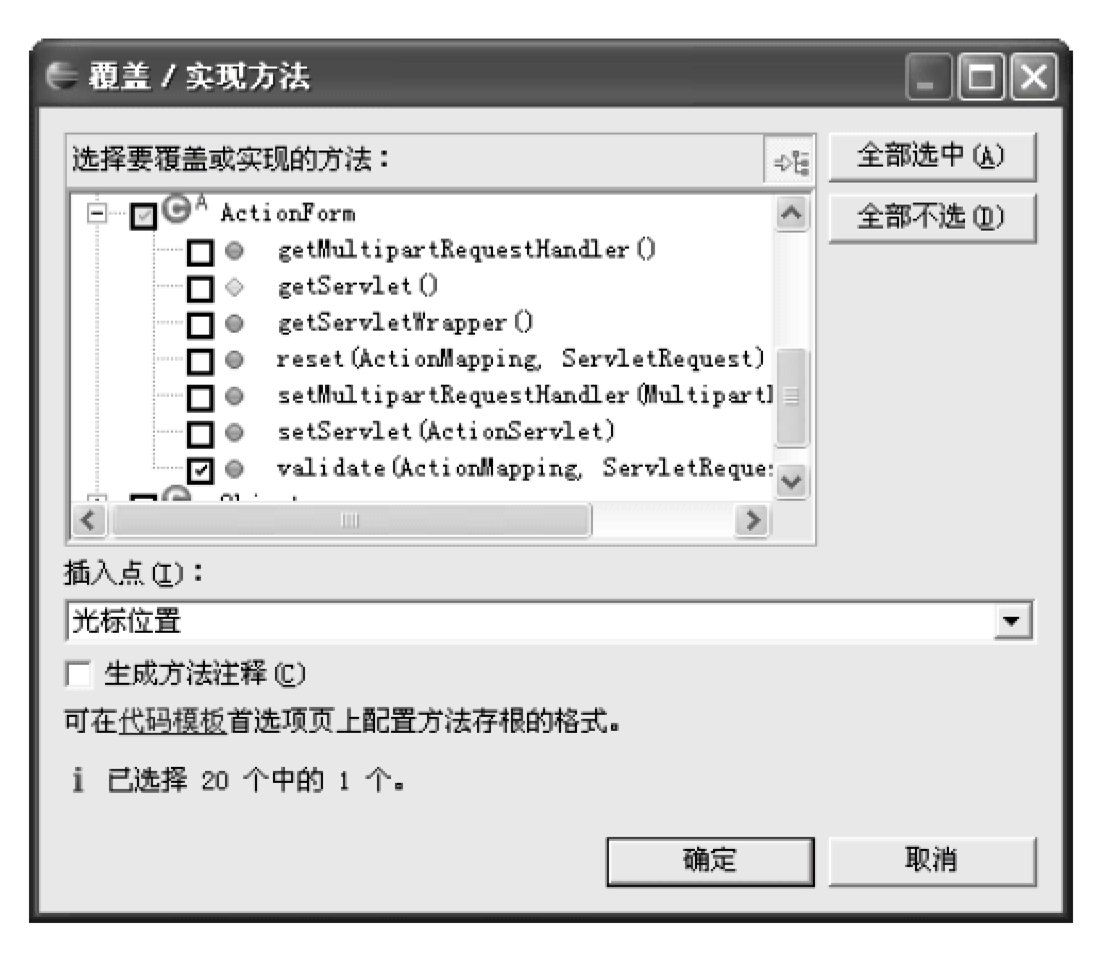


图 7-17　【覆盖/实现方法】窗口

（2）在【覆盖/实现方法】窗口中选择 validate 方法，单击【确定】按钮，生成如下代码：

```
    public ActionErrors validate(ActionMapping arg0, HttpServletRequest arg1) {

        return super.validate(arg0, arg1);
    }
```

（3）编辑该方法，输入如下代码：

```
public ActionErrors validate(ActionMapping arg0, HttpServletRequest arg1) {
        //声明 ActionErrors 对象，用来保存所有的错误信息
        ActionErrors errors = new ActionErrors();
        //如果留言的内容为空，就会返回错误信息
        if (content.length() < 1)
            errors.add("content", new ActionMessage("content is null!"));
        //如果作者的姓名为空，就会返回错误信息
        if (author.length() < 1)
            errors.add("author", new ActionMessage("author is not set!"));
        return errors;
    }
```

（4）将修改后的 MessageForm 类复制到 Tomcat 的相应路径下，重新启动 Tomcat，启动在线留言板应用程序。当输入的内容为空，或者作者的名字为空时就会跳转到提前设置好的错误页面上。其执行的数据验证流程如下：

① 用户在浏览器中输入 http://localhost:8080/OnlineBBS/publish.do。

② Tomcat 接到请求后在 web.xml 文件中查找<url-pattern>属性为“*.do”的<servlet-mapping>元素：

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
 </servlet-mapping>
```

③ 通过<servlet-name>属性指定名字“action”，在 web.xml 中查找其对应的 Servlet：

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
```

④ Tomcat 把请求转发给 org.apache.struts.action.ActionServlet 类，ActionServlet 根据用户请求的路径“/publish.do”，在配置文件中找到 path 属性为“/publish”的<action>元素。

```
<action
    path="/publish"
    forward="/pages/messageForm.jsp"/>
```

⑤ 页面跳转至 messageForm.jsp 页面。当用户在页面中输入留言内容和作者姓名后，提交这些数据，其请求路径为“input”。

```
<html:form action="input">
```

⑥ 在 Struts 配置页面中查询 path 属性值“/input”的<action>元素。

```
<action
     path="/input"
     type="MessageFormAction"
     name="messageForm"
     scope="request"
     validate="true"
     input="/pages/error.jsp"/>
```

⑦ ActionServlet 根据<action>元素的 name 属性，创建一个 MessageForm 对象，把客户端提交的数据传递给 MessageForm 对象，再把 MessageForm 对象保存在<action>元素的 scope 属性指定的 request 范围内。

⑧ 因为<action>元素的 validate 属性为 true，ActionServlet 就会调用 MessageForm 对象

的 validate 方法对数据进行验证。

```
ActionErrors errors = new ActionErrors();
		if (content.length() < 1)
			errors.add("content", new ActionMessage("content is null!"));
		if (author.length() < 1)
			errors.add("author", new ActionMessage("author is not set!"));
		return errors;
```

⑨ 根据<action>元素的 input 属性，把客户请求转发给 error.jsp 页面。

⑩ error.jsp 的<html:errors>标签从 request 范围内读取 ActionErrors 对象，从中取得 ActionMessage 对象，把它包含的错误信息显示在网页上。

第 8 章　Hibernate 开发实例——图书管理系统

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻量级的对象封装，使得 Java 程序员可以随心所欲地使用对象编程思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用，最具革命意义的是，Hibernate 可以在应用 EJB 的 J2EE 架构中取代 CMP，完成数据持久化的重任。

本章以图书管理系统为例详细介绍了在 Eclipse 中开发 Hibernate 实例的具体步骤，内容包括数据库的配置，创建持久化对象，生成 Hibernate 映射文件以及生成配置文件等内容，在本章的最后通过一个图书管理系统详细介绍了通过 Hibernate 进行数据库操作的详细步骤。

8.1　下载并安装 Hibernate Synchronizer 插件

Hibernate Synchronizer 是一个 Eclipse 插件，可以自动生成*.hbm 文件、持久化类和 DAO，大大降低开发 Hibernate 应用的难度。本节介绍如何下载和安装 Hibernate Synchronizer 插件。

JBoss Eclipse IDE 插件中包括 Hibernate Tools，按照 6.1 节介绍的步骤安装 JBoss Eclipse IDE 插件后即可完成 Hibernate Synchronizer 插件的安装，打开如图 8-1 所示的【关于 Eclipse SDK 插件】窗口，其中包含了 4 个 Hibernate 插件，说明插件安装成功。

关于 Eclipse SDK 插件

提供者	插件名	版本	插件标识
Eclipse.org	文本编辑器框架	3.1.1	org.eclipse.ui.workbench.te...
Eclipse.org	文件缓冲区	3.1.0	org.eclipse.core.filebuffers
Eclipse.org	小组支持核心	3.1.1	org.eclipse.team.core
Eclipse.org	小组支持用户界面	3.1.1	org.eclipse.team.ui
Eclipse.org	用于 Windows 的安装/更新...	3.1.0	org.eclipse.update.core.win32
Eclipse.org	用于 Windows 的标准窗口小...	3.1.1	org.eclipse.swt.win32.win32...
Eclipse.org	重构核心	3.1.0	org.eclipse.ltk.core.refact...
Eclipse.org	重构用户界面	3.1.1	org.eclipse.ltk.ui.refactoring
Eclipse.org	自动更新调度程序	3.1.0	org.eclipse.update.scheduler
Eclipse.org	浏览器支持	3.1.1	org.eclipse.ui.browser
Hibernate Team	Hibernate Console	3.1.0.beta4a	org.hibernate.eclipse.console
Hibernate Team	Hibernate Core Plugin	3.1.0.beta4a	org.hibernate.eclipse
Hibernate Team	Hibernate Help	3.1.0.beta4a	org.hibernate.eclipse.help
Hibernate Team	Mapper Plug-in	3.1.0.beta4a	org.hibernate.eclipse.mapper
JBoss, Inc	JBoss Eclipse IDE Help Pl...	1.5.1.GA	org.jboss.ide.eclipse.help
JBoss, Inc	JBossIDE First Run Plug-in	1.5.1.GA	org.jboss.ide.eclipse.firstrun
JBoss, Inc.	JBoss AOP Core Plug-in	1.1.1.GA	org.jboss.ide.eclipse.jdt.a...
JBoss, Inc.	JBoss AOP UI Plug-in	1.1.1.GA	org.jboss.ide.eclipse.jdt.a...
JBoss, Inc.	JBoss Core Plug-in	1.5.1.GA	org.jboss.ide.eclipse.core
JBoss, Inc.	JBoss Deployer Core Plug-in	1.5.1.GA	org.jboss.ide.eclipse.deplo...
JBoss, Inc.	JBoss Deployer UI Plug-in	1.5.1.GA	org.jboss.ide.eclipse.deplo...
JBoss, Inc.	JBoss Development Core Pl...	1.5.1.GA	org.jboss.ide.eclipse.jdt.core
JBoss, Inc.	JBoss Development J2EE Co...	1.5.1.GA	org.jboss.ide.eclipse.jdt.j...

更多信息(I)　　确定

图 8-1　【关于 Eclipse SDK 插件】窗口

8.2　图书管理系统需求分析

图书管理系统分为用户管理和图书管理两大部分，分别具有如下功能：

- 用户分为系统管理员、书籍管理员和借阅管理员 3 种角色，不同角色具有不同的权限。
- 用户登录和用户管理功能。
- 图书管理包括增加图书信息、删除图书信息和修改图书信息功能。
- 借书和还书管理，修改借书和还书记录信息。
- 查询所有书籍列表、书籍借阅情况和所有用户列表。

其运行界面如图 8-2 所示，整个系统分为系统管理、书籍管理、借书管理、还书管理和信息查询 5 大部分。



图 8-2　图书管理系统运行界面

系统管理完成对用户登录和用户权限的管理。用户权限分为“系统管理员”、“书籍管理员”和“借阅管理员”3 种。用户管理可以增加用户、修改用户信息和删除用户。

书籍管理完成对所有书籍信息的维护。分为“添加书籍”、“修改书籍”和“删除书籍”3 部分功能。借书管理完成对所有已出借图书信息的维护，分为“出借图书”和“修改出借图书信息”两部分功能。

8.3　配置数据库

本节在 SQL Server 2000 中创建图书管理系统的数据库，关于 SQL Server 数据库的相关知识可参见其相应文档。

跟我做

（1）打开 SQL Server 企业管理器，创建名称为“demo”的数据库，如图 8-3 所示。



图 8-3　新建 demo 数据库

（2）打开 SQL Server 的 SQL 查询分析器，选择默认数据库为刚才创建的“demo”数据库，输入如下 SQL 脚本：

```
//创建 books 表，保存所有有关书籍的信息
create table books (
BookName varchar(20),
Press varchar(20),
Author varchar(20),
address varchar(50),
PressDate datetime,
Price float,
Com varchar(20),
books_count int,
borrowed_count int,
constraint ID_Contraint_PK primary key ( BookName ));
//创建表 bookBrowse，保存所有书籍借阅情况信息
create table bookBrowse (
StudentName varchar(40),
BookName varchar(40),
ReturnDate datetime,
BorrowDate datetime,
Com varchar(40),
Is_Returned char(2),
constraint ID_BookBrowse_Containt primary key ( StudentName ));
//创建表 UserTable，保存所有的用户信息
create table UserTable(
UserName varchar(40),
Password varchar(40),
Power varchar(40),
constraint ID_User_Containt primary key ( UserName ));
```

（3）单击 ▶ 按钮，执行上述 SQL 脚本，生成 3 个表：books 表、bookBrowse 表和 UserTable 表，分别保存书籍信息、书籍出借信息和用户信息，如图 8-4～图 8-6 所示。

注意： 为了简单起见，books 表和 UserTable 表分别以书籍名和用户名作为主键，所以不允许出现重名的书籍和用户。

列名	数据类型	长度	允许空
BookName	varchar	20	
Press	varchar	20	✓
Author	varchar	20	✓
address	varchar	50	✓
PressDate	datetime	8	✓
Price	float	8	✓
Com	varchar	20	✓
books_count	int	4	✓
borrowed_count	int	4	✓

图 8-4　表 books

列名	数据类型	长度	允许空
StudentName	varchar	40	
BookName	varchar	40	✓
ReturnDate	datetime	8	✓
BorrowDate	datetime	8	✓
Com	varchar	40	✓
Is_Returned	char	2	✓

图 8-5　表 bookBrowse

列名	数据类型	长度	允许空
UserName	varchar	40	
Password	varchar	40	✓
Power	varchar	40	✓

图 8-6　表 UserTable

books 表记录书籍的书名、出版社、作者、地址、出版日期、价格、书籍数量和被借阅数量；bookBrowse 表记录学生姓名、书籍书名、借阅日期、应还日期和是否归还；UserTable 表中记录用户的姓名、密码和角色。

传统的对关系数据库表的访问都是通过 SQL 语句进行的，本章利用 Hibernate 框架来封装对关系数据库的访问，使得可以完全以面向对象的方式对上述表格进行读写操作，从而提高数据库开发效率。

8.4　生成配置文件 hibernate.cfg.xml

Hibernate 运行时需要获取一些底层实现的基本信息，包括数据库 URL、数据库用户、数据库用户密码、数据库 JDBC 驱动类和数据库 dialect 等。Hibernate 同时支持 xml 格式的配置文件，以及传统的 properties 文件配置方式。本章采用基于 xml 格式文件的配置方式，这些信息都包含在默认名称为 hibernate.cfg.xml 的文件中。本节介绍如何在 Eclipse 中快速生成 hibernate.cfg.xml 文件。

跟我做

（1）创建名称为“Library”的 Java 工程。单击【文件】菜单，选择【新建】|【其他】命令，打开如图 8-7 所示的【新建】对话框。

（2）选择【Hibernate Configuration File】选项，单击【下一步】按钮，在图 8-8 的【输入或选择父文件夹】文本框中选择“Library”工程，单击【下一步】按钮，打开数据库配置对话框。




图 8-7　【新建】对话框



图 8-8　选择工程名称

（3）在数据源配置对话框中输入如下数据库配置信息，如图 8-9 所示。

- Database dialect：SQLServer
- Driver class：com.microsoft.jdbc.sqlserver.SQLServerDriver
- Connection URL：jdbc:microsoft:sqlserver://localhost:1433;databaseName=demo
- Username：sa（根据实际配置）

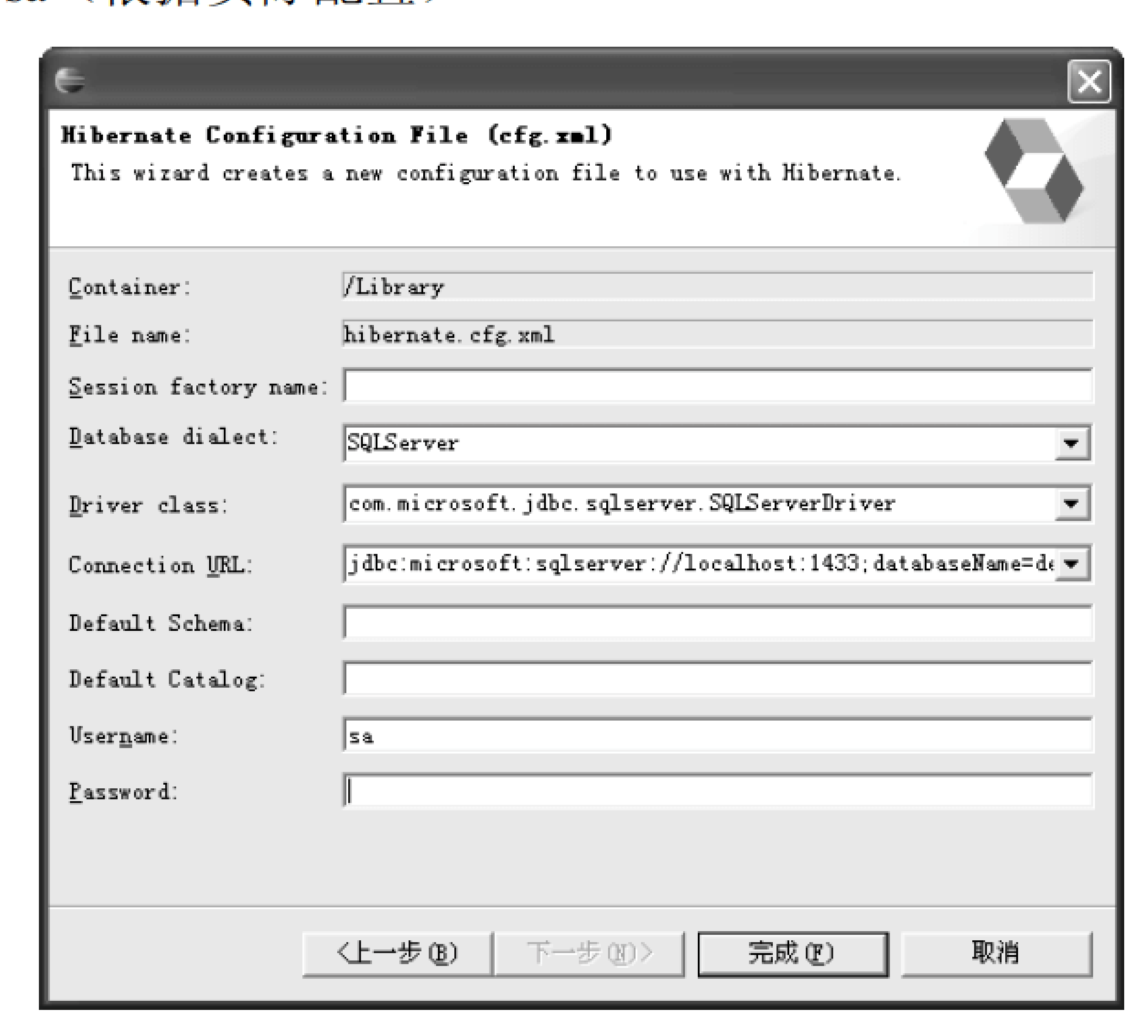


图 8-9　数据库配置窗对话框

（4）单击【完成】按钮，在 Library 工程的根目录下生成 hibernate.cfg.xml 文件，其内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
```

```
    <session-factory >
    <!--数据库 JDBC 驱动类-->
        <property
name="hibernate.connection.driver_class">com.microsoft.jdbc.sqlserver.SQLServerDriver</proper
ty>
    <!--数据库密码-->
        <property name="hibernate.connection.password"></property>
     <!--数据库的 URL-->
        <property
name="hibernate.connection.url">jdbc:microsoft:sqlserver://localhost:1433;databaseName=demo<
/property>
     <!--数据库的用户名-->
        <property name="hibernate.connection.username">sa</property>
     <!--每个数据库都有其对应的 Dialet 以匹配其平台特性-->
        <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>
    </session-factory>
</hibernate-configuration>
```

hibernate.cfg.xml 文件可以包含构建 SessionFactory 实例的所有配置信息。当使用如下代码

```
SessionFactory sessions=new Configuration().configure().buildSessionFactory();
```

初始化 Hibernate 时，Hibernate 会在 classpath 中寻找文件名为 hibernate.cfg.xml 的文件。

注意： 如果运行时出现如图 8-10 所示的错误信息，这是因为在配置文件中设置了 session-factory 的 name 属性，这样 hibernate 会试图把这个 sessionfactory 注册到 jndi 中去，从而报告错误。去掉 name 属性即可，所以在 hibernate.cfg.xml 中不要设置 session-factory 的 name 属性。

```
控制台 ×   任务 搜索
<已终止> Test [Java 应用程序] C:\Program Files\Java\jre1.5.0_06\bin\javaw.exe (2006-8-5 21:29:22)
2006-8-5 21:29:24 org.hibernate.util.NamingHelper getInitialContext
信息: JNDI InitialContext properties:{}
2006-8-5 21:29:24 org.hibernate.impl.SessionFactoryObjectFactory addInstance
警告: Could not bind factory to JNDI
javax.naming.NoInitialContextException: Need to specify class name in environment or system property, or as
	at javax.naming.spi.NamingManager.getInitialContext(Unknown Source)
	at javax.naming.InitialContext.getDefaultInitCtx(Unknown Source)
	at javax.naming.InitialContext.getURLOrDefaultInitCtx(Unknown Source)
	at javax.naming.InitialContext.getNameParser(Unknown Source)
	at org.hibernate.util.NamingHelper.bind(NamingHelper.java:52)
	at org.hibernate.impl.SessionFactoryObjectFactory.addInstance(SessionFactoryObjectFactory.java:90)
	at org.hibernate.impl.SessionFactoryImpl.<init>(SessionFactoryImpl.java:290)
	at org.hibernate.cfg.Configuration.buildSessionFactory(Configuration.java:1176)
	at library.main.HibernateUtil.<clinit>(HibernateUtil.java:18)
```

图 8-10　配置 session-factory name 属性后的出错信息

8.5　创建持久化对象

Hibernate 从本质上来讲是一种“对象－关系型数据映射”（Object Relational Mapping 简称 ORM）。POJO 在这里体现的就是 ORM 中 Object 层的语义，而映射（Mapping）文件则是将对象（Object）与关系型数据（Relational）相关联的纽带，在 Hibernate 中，映射文件通常以“.hbm.xml”作为后缀。

8.5.1　生成映射文件和持久化对象

本小节介绍如何在 Eclipse 中根据数据库中的表结构生成映射文件和持久化对象。通过 Hibernate Synchronizer 插件可以方便地生成映射文件和持久化对象，方便 Hibernate 应用的开发。

跟我做

（1）单击 Eclipse 的【窗口】菜单，选择【打开透视图】|【其他】命令，打开【选择透视图】对话框，如图 8-11 所示。选择【Hibernate Console】选项，单击【确定】按钮，打开 Hibernate Console 透视图。



图 8-11　【选择透视图】对话框

Hibernate Console 透视图包括 Hibernate Configuration、Hibernate Dynamic Query Translator、Hibernate Entity Model 和 Hibernate Query Result 等几个视图。

（2）右击 Hibernate Configuration 视图的空白区域，在快捷菜单中选择【Add configuration】命令，打开【Create Hibernate Console Configuration】对话框，如图 8-12 所示。

（3）单击【Configuration file】文本框右侧的 Browse... 按钮并选择“\Library\hibernate. cfg.xml”；单击“Classpath”组中的 Add JAR/Dir... 按钮将 SQL Server 数据库的 JDBC 驱动类 msbase.jar、mssqlserver.jar 等加入到 classpath 中；在【Name】文本框中输入“Library”，单击【完成】按钮，创建名称为“Library”的配置，如图 8-13 所示。

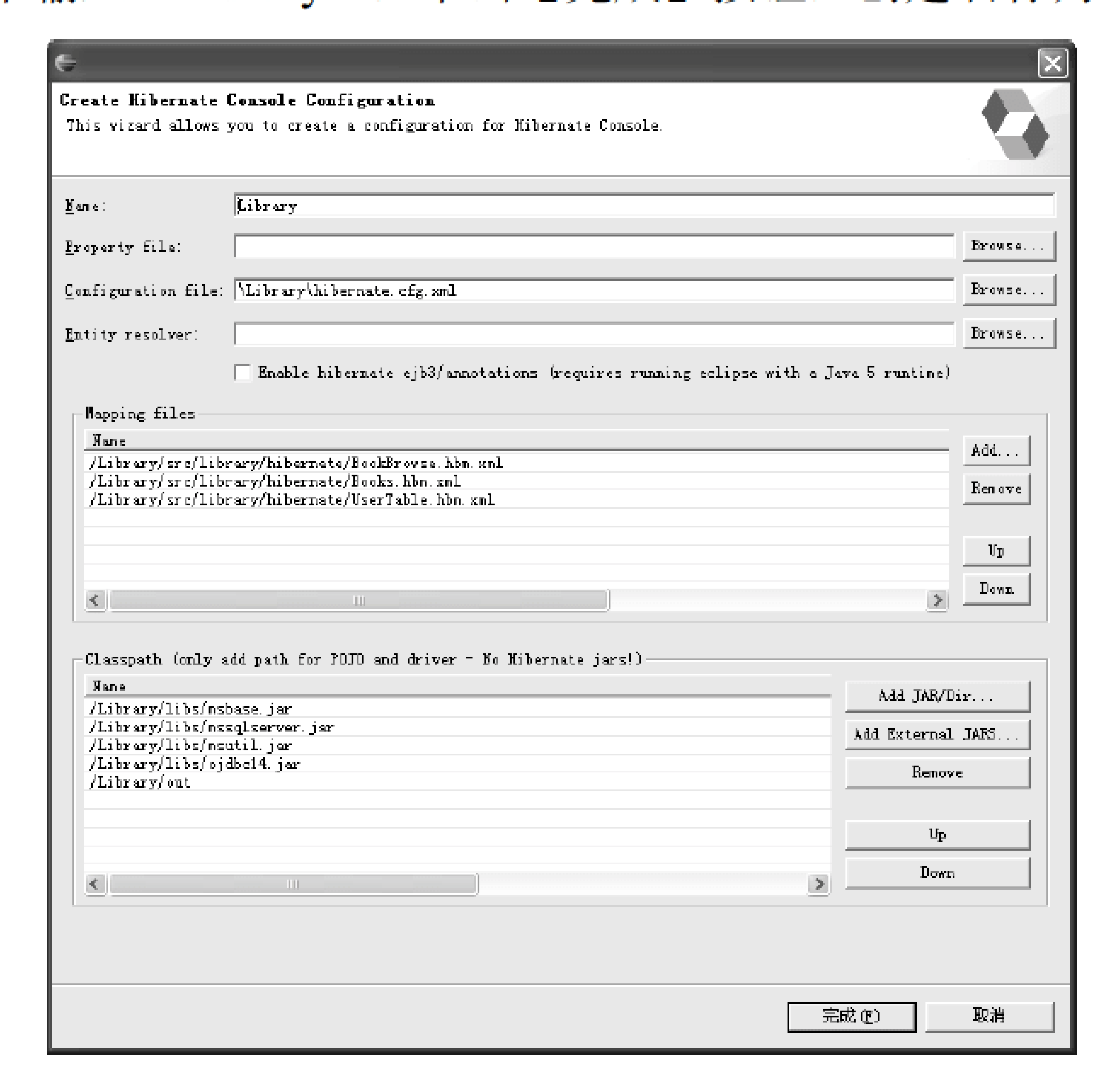


图 8-12　生成 hibernate Console Configuration

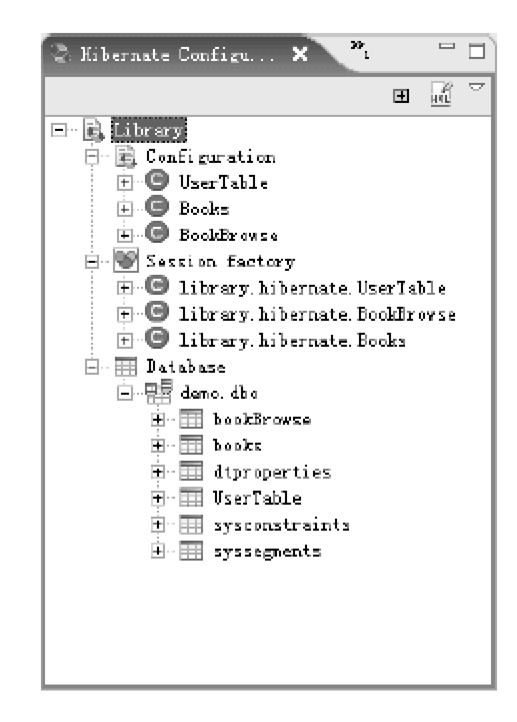


图 8-13　Hibernate Configuration 视图

（4）单击工具栏中的 按钮右边的下拉箭头，在下拉菜单中选择【Hibernate Code Generation】命令，打开如图 8-14 所示的【Hibernate Code Generation】对话框。

（5）在【Hibernate Code Generation】对话框中的【Main】选项卡中输入如下配置信息：

- ❑ 名称：codegeneration
- ❑ Console configuration：Library
- ❑ Output directory：\Library\src
- ❑ Package：library.hibernate

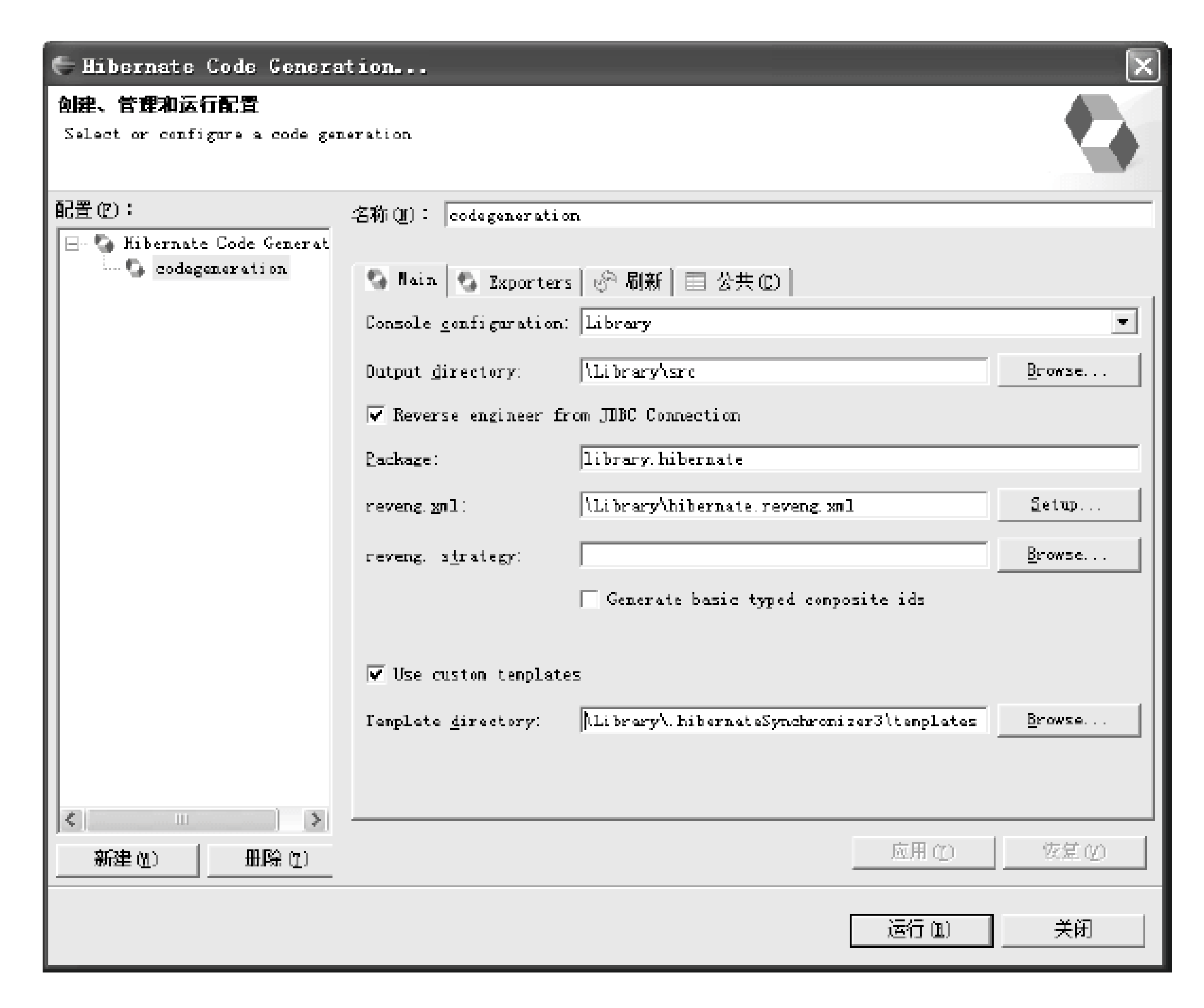


图 8-14　【Hibernate Code Generation】对话框

创建名称为“codegeneration”的 Hibernate Code Generation 配置，生成的目标代码存放在“\library\src”目录下，其包名为“library.hibernate”。

（6）单击【Exporters】选项卡选中如下各项，如图 8-15 所示。

- ❑ Generate domain code(.java)
- ❑ JDK1.5 Constructs(generics,etc.)
- ❑ Generate DAO code(.java)
- ❑ Generate mappings(hbm.xml)

（7）单击【运行】按钮，在 Library 工程的“library.hibernate”包中生成如图 8-16 所示的文件。





图 8-15　Exporters 配置

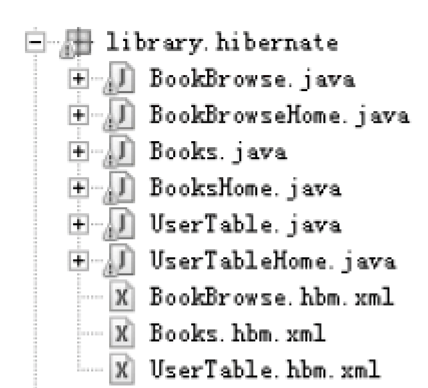


图 8-16　生成的映射文件和持久化对象

8.5.2 对持久化对象的分析

BookBrowse.java、Books.java 和 UserTable.java 3 个生成的类是标准的 JavaBean，Hibernate 插件根据数据库的表结构自动生成了 3 个持久化对象，对于每个属性都有其对应的 getter/setter 方法。

注意： 这 3 个类中的英文注释是 Hibernate 插件自动生成的，为了便于读者的理解，这里加上了部分中文注释。

（1）BookBrowse.java 文件是标准的 JavaBean 对应于 demo 数据库中的 bookbrowse 表，包括学生姓名、书名、归还日期、借阅日期、备注和是否归还等信息。其代码如下所示：

```
package library.hibernate;
// Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3

import java.util.Date;
/**
 * BookBrowse generated by hbm2java
 */

public class BookBrowse  implements java.io.Serializable {


    // Fields
     //学生名字
     private String studentName;
     //书名
     private String bookName;
     //归还日期
     private Date returnDate;
     //借阅日期
     private Date borrowDate;
     //备注，评论信息
     private String com;
     //是否归还
     private String isReturned;


    // Constructors

    /** 默认构造函数 */
    public BookBrowse() {
    }

     /** minimal constructor */
    public BookBrowse(String studentName) {
        this.studentName = studentName;
```

```
    }

    /** full constructor */
    public BookBrowse(String studentName, String bookName, Date returnDate, Date borrowDate,
String com, String isReturned) {
        this.studentName = studentName;
        this.bookName = bookName;
        this.returnDate = returnDate;
        this.borrowDate = borrowDate;
        this.com = com;
        this.isReturned = isReturned;
    }

    // Property accessors 取得学生姓名

    public String getStudentName() {
        return this.studentName;
    }
    //设置学生姓名
    public void setStudentName(String studentName) {
        this.studentName = studentName;
    }
    //取得书名字段的值
    public String getBookName() {
        return this.bookName;
    }
    //设置书名字段
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }
    //取得归还日期字段
    public Date getReturnDate() {
        return this.returnDate;
    }
    //设置归还日期字段
    public void setReturnDate(Date returnDate) {
        this.returnDate = returnDate;
    }
    //取得借阅日期
    public Date getBorrowDate() {
        return this.borrowDate;
    }
    //设置借阅日期
    public void setBorrowDate(Date borrowDate) {
        this.borrowDate = borrowDate;
    }
    //取得备注信息
    public String getCom() {
        return this.com;
```

```
    }
    //设置备注信息
    public void setCom(String com) {
        this.com = com;
    }
    //取得是否归还状态
    public String getIsReturned() {
        return this.isReturned;
    }
    //设置是否归还状态
    public void setIsReturned(String isReturned) {
        this.isReturned = isReturned;
    }
}
```

（2）Books.java 类对应于 demo 数据库中的 books 表，是标准的 JavaBean，包括书名、出版社、作者、地址、出版日期和价格等相关信息。其代码如下所示：

```
package library.hibernate;
// Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3

import java.util.Date;

/**
 * Books generated by hbm2java
 */

public class Books  implements java.io.Serializable {


    // Fields
     //书名
     private String bookName;
     //出版社
     private String press;
     //作者
     private String author;
     //地址
     private String address;
     //出版日期
     private Date pressDate;
     //价格
     private Double price;
     //备注
     private String com;
     //图书数量
     private Integer booksCount;
     //已借阅数量
     private Integer borrowedCount;
```

```
    // Constructors

    /** default constructor */
    public Books() {
    }

     /** minimal constructor */
    public Books(String bookName) {
        this.bookName = bookName;
    }

    /** full constructor */
    public Books(String bookName, String press, String author, String address, Date pressDate,
Double price, String com, Integer booksCount, Integer borrowedCount) {
        this.bookName = bookName;
        this.press = press;
        this.author = author;
        this.address = address;
        this.pressDate = pressDate;
        this.price = price;
        this.com = com;
        this.booksCount = booksCount;
        this.borrowedCount = borrowedCount;
    }



    // Property accessors
    //取得书名属性的值
    public String getBookName() {
        return this.bookName;
    }
    //设置书名属性
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }
    //取得出版社字段的值
    public String getPress() {
        return this.press;
    }
    //设置出版社字段
    public void setPress(String press) {
        this.press = press;
    }
    //取得作者信息
    public String getAuthor() {
        return this.author;
    }
    //设置作者信息
```

```
public void setAuthor(String author) {
	this.author = author;
}
//取得地址信息
public String getAddress() {
	return this.address;
}
//设置地址信息
public void setAddress(String address) {
	this.address = address;
}
//取得出版日期
public Date getPressDate() {
	return this.pressDate;
}
//设置出版日期
public void setPressDate(Date pressDate) {
	this.pressDate = pressDate;
}
//取得价格信息
public Double getPrice() {
	return this.price;
}
//设置价格信息
public void setPrice(Double price) {
	this.price = price;
}
//取得备注信息
public String getCom() {
	return this.com;
}
//设置备注信息
public void setCom(String com) {
	this.com = com;
}
//取得图书数量
public Integer getBooksCount() {
	return this.booksCount;
}
//设置图书数量
public void setBooksCount(Integer booksCount) {
	this.booksCount = booksCount;
}
//取得已借阅图书数量
public Integer getBorrowedCount() {
	return this.borrowedCount;
}
//设置已借阅图书数量
public void setBorrowedCount(Integer borrowedCount) {
```

```
            this.borrowedCount = borrowedCount;
      }
}
```

（3）UserTable.java 类对应于 demo 数据库中的 usertable 表，包括用户名、密码和用户权限 3 个属性，是标准的 JavaBean。其代码如下所示：

```
package library.hibernate;
// Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3

/**
 * UserTable generated by hbm2java
 */

public class UserTable   implements java.io.Serializable {
      // Fields
       //用户名
       private String userName;
       //密码
       private String password;
       //用户权限
       private String power;

      // Constructors

      /** default constructor */
      public UserTable() {
      }

       /** minimal constructor */
      public UserTable(String userName) {
            this.userName = userName;
      }

      /** full constructor */
      public UserTable(String userName, String password, String power) {
            this.userName = userName;
            this.password = password;
            this.power = power;
      }



      // Property accessors
      //取得用户名字段
      public String getUserName() {
            return this.userName;
      }
      //设置用户名字段
      public void setUserName(String userName) {
```

```
		this.userName = userName;
	}
	//取得密码字段
	public String getPassword() {
		return this.password;
	}
	//设置密码字段
	public void setPassword(String password) {
		this.password = password;
	}
	//取得用户权限
	public String getPower() {
		return this.power;
	}
	//设置用户权限
	public void setPower(String power) {
		this.power = power;
	}


}
```

8.6　创建映射文件

Hibernate 之所以能够判断实体类和数据表之间的对应关系，是因为有 XML 映射文件。通过 Hibernate Code Generation 既可以生成持久化对象也可以生成映射文件。本节介绍在 8.5 节中生成的 BookBrowse.hbm.xml、Books.hbm.xml、UserTable.hbm.xml 3 个映射文件。

BookBrowse.hbm.xml 描述了 BookBrowse 类和 bookbrowse 表之间的映射关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3 -->
<hibernate-mapping>
	<!--library.hibernate.BookBrowse 类和 bookbrowse 表对应关系描述-->
	<classname="library.hibernate.BookBrowse"table="bookBrowse"schema="dbo"catalog=
"demo">
		<!--studentName 属性对应 StudentName 字段-->
		<id name="studentName" type="string">
			<column name="StudentName" length="40" />
			<generator class="assigned" />
		</id>
		<!--bookName 属性对应 BookName 字段-->
		<property name="bookName" type="string">
			<column name="BookName" length="40" />
```

```
        </property>
        <!--returnDate 属性对应 ReturnDate 字段-->
        <property name="returnDate" type="timestamp">
            <column name="ReturnDate" length="23" />
        </property>
         <!-- borrowDate 属性对应 BorrowDate 字段-->
        <property name="borrowDate" type="timestamp">
            <column name="BorrowDate" length="23" />
        </property>
         <!-- com 属性对应 Com 字段-->
        <property name="com" type="string">
            <column name="Com" length="40" />
        </property>
         <!-- isReturned 属性对应 Is_Returned 字段-->
        <property name="isReturned" type="string">
            <column name="Is_Returned" length="2" />
        </property>
    </class>
</hibernate-mapping>
```

Books.hbm.xml 描述了 Books 类和 books 表之间的映射关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3 -->
<hibernate-mapping>
    <!--library.hibernate.Books 类和 books 表对应关系描述-->
    <class name="library.hibernate.Books" table="books" schema="dbo" catalog="demo">
         <!-- bookName 属性对应 BookName 字段-->
        <id name="bookName" type="string">
            <column name="BookName" length="20" />
            <generator class="assigned" />
        </id>
        <!-- press 属性对应 Press 字段-->
        <property name="press" type="string">
            <column name="Press" length="20" />
        </property>
        <!-- author 属性对应 Author 字段-->
        <property name="author" type="string">
            <column name="Author" length="20" />
        </property>
        <!-- address 属性对应 address 字段-->
        <property name="address" type="string">
            <column name="address" length="50" />
        </property>
        <!-- pressDate 属性对应 PressDate 字段-->
        <property name="pressDate" type="timestamp">
            <column name="PressDate" length="23" />
        </property>
```

```
        <!-- price 属性对应 Price 字段-->
        <property name="price" type="java.lang.Double">
            <column name="Price" precision="53" scale="0" />
        </property>
        <!-- com 属性对应 Com 字段-->
        <property name="com" type="string">
            <column name="Com" length="20" />
        </property>
        <!-- booksCount 属性对应 books_count 字段-->
        <property name="booksCount" type="java.lang.Integer">
            <column name="books_count" />
        </property>
        <!-- borrowedCount 属性对应 borrowed_count 字段-->
        <property name="borrowedCount" type="java.lang.Integer">
            <column name="borrowed_count" />
        </property>
    </class>
</hibernate-mapping>
```

UserTable.hbm.xml 描述了 UserTable 类和 usertable 表之间的对应关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 2006-8-5 16:17:23 by Hibernate Tools 3.1.0 beta3 -->
<hibernate-mapping>
    <!--library.hibernate.UserTable 类和 usertable 表对应关系描述-->
    <class name="library.hibernate.UserTable" table="UserTable" schema="dbo"catalog="demo">
        <!-- userName 属性对应 UserName 字段-->
        <id name="userName" type="string">
            <column name="UserName" length="40" />
            <generator class="assigned" />
        </id>
        <!-- password 属性对应 Password 字段-->
        <property name="password" type="string">
            <column name="Password" length="40" />
        </property>
        <!-- power 属性对应 Power 字段-->
        <property name="power" type="string">
            <column name="Power" length="40" />
        </property>
    </class>
</hibernate-mapping>
```

</class>定义实体类和数据表之间的关系，name 属性 library.hibernate.UserTable 是实体类（用类全名）的名字，UserTable 是对应的数据表。</id>定义了主键 id 字段所用的键值生成方法。</property>定义了实体类和表字段的关联，name 属性定义了类的字段，column 属

性定义了数据库表字段名。Hibernate 通过这里的设置建立起实体类和数据库表之间的对应关系。

8.7　Hibernate 操作数据库的方法

通过 Hibernate 可以简化对数据库的操作，本节首先创建一个 HibernateUtil 类，用于管理 session，然后介绍如何通过 Hibernate 实现数据库的查询、插入、删除和更新操作。

SessionFactory 用来创建 Session 实例，通过 Configuration 实例构建 SessionFactory。Configuration 实例根据当前的配置信息，构造 SessionFactory 实例并返回。一旦 SessionFactory 构造完毕，即被赋予特定的配置信息。

Session 是持久层操作的基础，相当于 JDBC 的 Connection。通过 SessionFactory 实例构建。Session 实例提供的 saveOrUpdate、delete 和 createQuery 方法分别实现了数据库的插入更新、删除和查询操作，简化了数据库的基本操作。

跟我做

（1）在“Library”工程的“src”文件夹中创建“library.main”包，在“library.main”包中创建 HibernateUtil.java 文件，并输入如下内容：

```
package library.main;
import java.io.File;
import org.hibernate.*;
import org.hibernate.cfg.*;

public class HibernateUtil {
	private static final SessionFactory sessionFactory;
	static {
		try {
			//hibernate.cfg.xml 文件
			File file = new File(
					"E:\\Eclipsebook\\eclipse\\workspace\\Library\\src\\hibernate.cfg.xml");
			//根据 hibernate.cfg.xml 中的配置信息创建 SessionFactory
			sessionFactory = new Configuration().configure(file)
					.buildSessionFactory();
		} catch (Throwable ex) {
			//创建 SessionFactory 失败信息
			System.err.println("Initial SessionFactory creation failed." + ex);
			throw new ExceptionInInitializerError(ex);
		}
	}
	//得到 SessionFactory 的静态方法
	public static SessionFactory getSessionFactory() {
		return sessionFactory;
	}
}
```

（2）在“Library”工程的“src”文件夹中创建“library.test”包，创建 Test.java 文件，在该 Test.java 文件中测试数据库的插入、更新、删除和查询操作。

（3）插入和更新数据库的基本操作。

```
//取得 SessionFactory 实例
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
//打开一个 Session 实例
 Session session = sessionFactory.openSession();
//开始事务
 Transaction tx = session.beginTransaction();
 //创建 UserTable 类实例
  UserTable userTable=new UserTable();
//设置 userName 属性
  userTable.setUserName("张三");
//设置 password 属性
  userTable.setPassword("123456");
//设置 power 属性
  userTable.setPower("图书管理员");
//插入和更新数据库
  session.saveOrUpdate(userTable);
//提交事务
 tx.commit();
//关闭会话
 session.close();
```

（4）从数据库中删除记录的基本操作。

```
//取得 SessionFactory 实例
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
//打开一个 Session 实例
 Session session = sessionFactory.openSession();
//开始事务
 Transaction tx = session.beginTransaction();
//创建 UserTable 类实例
 UserTable userTable=new UserTable();
//设置 userName 属性
 userTable.setUserName("张三");
//设置 password 属性
 userTable.setPassword("123456");
//设置 power 属性
 userTable.setPower("图书管理员");
 //  删除操作
 // session.delete(userTable);
//提交事务
 tx.commit();
 //关闭会话
session.close();
```

（5）查询数据库的基本操作。

```
        //取得 SessionFactory 实例
         SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        //打开一个 Session 实例
         Session session = sessionFactory.openSession();
        //开始事务
         Transaction tx = session.beginTransaction();
         //  查询 hql 语句
         String hql = "from UserTable where UserName='张三' and Password='123456'";
        //执行查询，查询结果为 Query 实例
         Query userList = session.createQuery(hql);
        //将查询结果放到一个 list 中
         List list = userList.list();
        //提交事务
         tx.commit();
        //关闭 session
         session.close();
```

8.8 系统主界面

系统主界面是整个系统的入口，所有系统功能都是通过主界面来实现的。主界面的运行效果图如图 8-2 所示，本节实现该主界面。

8.8.1 主界面窗体的创建

本小节将利用 Swing 中的图形控件生成主界面的窗体、菜单等内容，并且为每个菜单项添加了事件监听者。关于 Swing 的相关知识超出了本书介绍的范围，读者可以查看相关资料，本小节假设读者对 Swing 有一定的了解。

跟我做

（1）在“Library”工程的“src”文件夹中创建“library.main”包，创建 LibraryWindow.java 文件，继承 Jframe 类，并实现 ActionListener 接口。该文件是整个图书管理系统的入口，完成界面菜单的生成和菜单动作的响应等功能。

（2）编辑 LibraryWindow.java 文件，在其构造函数中输入如下代码：

```
    public LibraryWindow() {
         super("图书管理系统");
         // --系统管理菜单--
         menuBar = new JMenuBar();
         systemMenu = new JMenu("系统管理");
         // --用户管理菜单--
        userMGRMenu = new JMenu("用户管理");
        // --用户登录菜单--
         userLoginMenuItem = new JMenuItem("用户登录");
        // --添加用户菜单--
```

```
userAddMenuItem = new JMenuItem("添加用户");
// --修改用户菜单--
userModifyMenuItem = new JMenuItem("修改用户");
//删除用户菜单
userDeleteMenuItem = new JMenuItem("删除用户");
//退出菜单
exitMenuItem = new JMenuItem("退出");
// --将“用户登录”子菜单项加入到“系统菜单”中--
systemMenu.add(userLoginMenuItem);
// --将“添加用户”子菜单项加入到“用户管理”菜单中--
userMGRMenu.add(userAddMenuItem);
// --将“修改用户”子菜单项加入到“用户管理”菜单中--
userMGRMenu.add(userModifyMenuItem);
 // --将“删除用户”子菜单项加入到“用户管理”菜单中--
userMGRMenu.add(userDeleteMenuItem);
 // --将“用户管理”子菜单项加入到“系统管理”菜单中--
systemMenu.add(userMGRMenu);
 // --将“退出”子菜单项加入到“系统管理”菜单中--
systemMenu.add(exitMenuItem);
 // --为“用户登录”菜单项添加动作监听者--
userLoginMenuItem.addActionListener(this);
 // --为“添加用户”菜单项添加动作监听者--
userAddMenuItem.addActionListener(this);
 // --为“修改用户”菜单项添加动作监听者--
userModifyMenuItem.addActionListener(this);
 // --为“删除用户”菜单项添加动作监听者--
userDeleteMenuItem.addActionListener(this);
 // --为“退出”菜单项添加动作监听者--
exitMenuItem.addActionListener(this);
 // --将“系统管理”菜单项加入到菜单栏上--
menuBar.add(systemMenu);
// ---书籍管理菜单--
bookMGRMenu = new JMenu("书籍管理");
 // -- “添加书籍”菜单--
bookAddMenuItem = new JMenuItem("添加书籍");
// -- “修改书籍”菜单--
bookModifyMenuItem = new JMenuItem("修改书籍");
// -- “删除书籍”菜单--
bookDeleteMenuItem = new JMenuItem("删除书籍");
// --将“添加书籍”菜单加入到“书籍管理”菜单项中--
bookMGRMenu.add(bookAddMenuItem);
// --将“修改书籍”菜单加入到“书籍管理”菜单项中--
bookMGRMenu.add(bookModifyMenuItem);
// --将“删除书籍”菜单加入到“书籍管理”菜单项中--
bookMGRMenu.add(bookDeleteMenuItem);
// --将“添加书籍”菜单添加事件监听者--
bookAddMenuItem.addActionListener(this);
// --将“修改书籍”菜单添加事件监听者--
bookModifyMenuItem.addActionListener(this);
```

```
// --将“删除书籍”菜单添加事件监听者--
bookDeleteMenuItem.addActionListener(this);
//将“书籍管理”菜单添加到菜单栏中
menuBar.add(bookMGRMenu);
// --借书管理菜单--
borrowBookMenu = new JMenu("借书管理");
//“书籍出借”菜单
borrowBookMenuItem = new JMenuItem("书籍出借");
//“出借信息修改“菜单
borrowInfoMenuItem = new JMenuItem("出借信息修改");
//将“书籍出借”菜单加入到“书籍管理”菜单中
borrowBookMenu.add(borrowBookMenuItem);
//将“出借信息修改”菜单加入到“书籍管理”菜单中
borrowBookMenu.add(borrowInfoMenuItem);
//为“书籍出借”菜单添加事件监听者
borrowBookMenuItem.addActionListener(this);
//为“出借信息修改”菜单添加事件监听者
borrowInfoMenuItem.addActionListener(this);
//将“借书管理”菜单加入到菜单栏中
menuBar.add(borrowBookMenu);
// --还书管理菜单--
returnBookMenu = new JMenu("还书管理");
//“书籍还入”菜单
returnBookMenuItem = new JMenuItem("书籍还入");
//“书籍还入信息修改”菜单
returnInfoMenuItem = new JMenuItem("书籍还入信息修改");
//将“书籍还入”菜单加入到“还书管理”菜单中
returnBookMenu.add(returnBookMenuItem);
//将“书籍还入信息修改”菜单加入到“还书管理”菜单中
returnBookMenu.add(returnInfoMenuItem);
//为“书籍还入”菜单添加事件监听者
returnBookMenuItem.addActionListener(this);
//为“书籍还入信息修改”菜单添加事件监听者
returnInfoMenuItem.addActionListener(this);
//将“还书管理”菜单加入到菜单栏中
menuBar.add(returnBookMenu);
// --信息一览菜单--
infoBrowseMenu = new JMenu("信息查询");
//"书籍列表"菜单
bookListMenuItem = new JMenuItem("书籍列表");
//“借阅情况表”菜单
borrowBookListMenuItem = new JMenuItem("借阅情况表");
//“用户列表”菜单
userListMenuItem = new JMenuItem("用户列表");
//将“书籍列表”菜单添加到“信息一览”菜单中
infoBrowseMenu.add(bookListMenuItem);
//将“出借书籍列表”菜单添加到“信息一览”菜单中
infoBrowseMenu.add(borrowBookListMenuItem);
//将“用户列表”菜单添加到“信息一览”菜单中
```

```
            infoBrowseMenu.add(userListMenuItem);
            //为“书籍列表”菜单添加事件监听者
            bookListMenuItem.addActionListener(this);
            //为“出借图书列表”菜单添加事件监听者
            borrowBookListMenuItem.addActionListener(this);
            //为“用户列表”菜单添加事件监听者
            userListMenuItem.addActionListener(this);
            //将“信息一览”菜单加入到菜单栏中
            menuBar.add(infoBrowseMenu);
            //为 Window 添加菜单栏
            setJMenuBar(menuBar);
            //图片 Label
            titleLabel = new JLabel(new ImageIcon(".\\pic.jpg"));
            container = getContentPane();
            container.setLayout(new BorderLayout());
            panel1 = new JPanel();
            panel1.setLayout(new BorderLayout());
            panel1.add(titleLabel, BorderLayout.CENTER);
            container.add(panel1, BorderLayout.CENTER);
            setBounds(100, 50, 400, 300);
            show();
            // --设置初始功能:--
            userMGRMenu.setEnabled(false);
            bookMGRMenu.setEnabled(false);
            borrowBookMenu.setEnabled(false);
            returnBookMenu.setEnabled(false);
            infoBrowseMenu.setEnabled(false);
      }
```

生成了系统管理、书籍管理、借书管理、还书管理和信息一览 5 个菜单项，并且为其子菜单项添加了事件监听者。

8.8.2　为每个菜单项添加响应事件

本小节将在 8.8.1 节的基础上，为主界面的每个菜单项添加事件响应函数，并且控制每个新创建窗口的显示位置。在 actionPerformed（ActionEvent）方法中通过参数 ActionEvent 对象的 getActionCommand()方法可以得到用户点击的菜单项，然后进行不同的处理。

跟我做

（1）右击 LibraryWindow 类的任意空白区域，在快捷菜单中选择【源代码】|【覆盖/实现方法】命令，打开【覆盖/实现方法】对话框，选择“actionPerformed（ActionEvent）”，单击【确定】按钮，生成 actionPerformed 方法。

（2）编辑 LibraryWindow.java 文件，在 actionPerformed 方法中输入如下代码：

```
      // --设置每个菜单点击后出现的窗口和窗口显示的位置--
      public void actionPerformed(ActionEvent e) {
```

```
    //“用户登录”菜单的响应函数
    if (e.getActionCommand() == "用户登录") {
        //创建用户登录窗口
        UserLogin userLoginFrame = new UserLogin(this);
        Dimension frameSize = userLoginFrame.getPreferredSize();
        Dimension mainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的位置和大小
        userLoginFrame.setLocation((mainFrameSize.width - frameSize.width)
                / 2 + loc.x, (mainFrameSize.height - frameSize.height) / 2
                + loc.y);
        userLoginFrame.pack();
        userLoginFrame.show();
    } else if (e.getActionCommand() == "添加用户") {
        //“添加用户”菜单的响应函数
        UserAdd UserAddFrame = new UserAdd();
        Dimension FrameSize = UserAddFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的大小和位置
        UserAddFrame.setLocation((MainFrameSize.width - FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
        UserAddFrame.pack();
        UserAddFrame.show();
    } else if (e.getActionCommand() == "修改用户") {
        //“修改用户”菜单的响应函数
        UserModify UserModifyFrame = new UserModify();
        Dimension FrameSize = UserModifyFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的大小和位置
        UserModifyFrame.setLocation((MainFrameSize.width - FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
        UserModifyFrame.pack();
        UserModifyFrame.show();
    } else if (e.getActionCommand() == "删除用户") {
        //初始化“删除用户“窗口
        UserDelete UserDeleteFrame = new UserDelete();
        Dimension FrameSize = UserDeleteFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的大小和位置
        UserDeleteFrame.setLocation((MainFrameSize.width - FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
        UserDeleteFrame.pack();
        //显示窗口
```

```
            UserDeleteFrame.show();
        } else if (e.getActionCommand() == "添加书籍") {
            // "添加书籍" 窗口
            BookAdd BookAddFrame = new BookAdd();
            Dimension FrameSize = BookAddFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
           //设置窗口的大小和位置
            BookAddFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BookAddFrame.pack();
            //显示添加书籍窗口
            BookAddFrame.show();
        } else if (e.getActionCommand() == "修改书籍") {
            //初始化"修改书籍"窗口
            BookModify BookModifyFrame = new BookModify();
            Dimension FrameSize = BookModifyFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
            //设置窗口的大小和位置
            BookModifyFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BookModifyFrame.pack();
            //显示"修改书籍"窗口
            BookModifyFrame.show();
        } else if (e.getActionCommand() == "删除书籍") {
            //初始化"删除书籍"窗口
            BookDelete BookDeleteFrame = new BookDelete();
            Dimension FrameSize = BookDeleteFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
            //设置窗口的大小和位置
            BookDeleteFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BookDeleteFrame.pack();
            //显示窗口
            BookDeleteFrame.show();
        } else if (e.getActionCommand() == "书籍出借") {
            //初始化"书籍出借"窗口
            BorrowBook BorrowBookFrame = new BorrowBook();
            Dimension FrameSize = BorrowBookFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
            //设置窗口的大小和位置
            BorrowBookFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
```

```
                + loc.y);
        BorrowBookFrame.pack();
        //显示窗口
        BorrowBookFrame.show();
} else if (e.getActionCommand() == "出借信息修改") {
        //初始化“出借信息修改”窗口
        BorrowInfo BorrowInfoFrame = new BorrowInfo();
        Dimension FrameSize = BorrowInfoFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的大小和位置
        BorrowInfoFrame.setLocation((MainFrameSize.width - FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
        BorrowInfoFrame.pack();
        //显示窗口
        BorrowInfoFrame.show();
} else if (e.getActionCommand() == "书籍还入") {
        //初始化“书籍还入”窗口
        ReturnBook returnBookFrame = new ReturnBook();
        Dimension FrameSize = returnBookFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的大小和位置
        returnBookFrame.setLocation((MainFrameSize.width - FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
        returnBookFrame.pack();
        //显示窗口
        returnBookFrame.show();
} else if (e.getActionCommand() == "书籍还入信息修改") {
        //初始化“书籍还入信息修改”窗口
        ReturnInfo returnInfoFrame = new ReturnInfo();
        Dimension FrameSize = returnInfoFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
        //设置窗口的大小和位置
        returnInfoFrame.setLocation((MainFrameSize.width - FrameSize.width)
                / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                + loc.y);
        returnInfoFrame.pack();
        //显示窗口
        returnInfoFrame.show();
} else if (e.getActionCommand() == "书籍列表") {
        //初始化“书籍列表”窗口
        BookList BookListFrame = new BookList();
        Dimension FrameSize = BookListFrame.getPreferredSize();
        Dimension MainFrameSize = getSize();
        Point loc = getLocation();
```

```
            //设置窗口的大小和位置
            BookListFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            BookListFrame.pack();
            //显示窗口
            BookListFrame.show();
        } else if (e.getActionCommand() == "借阅情况表") {
            //初始化“借阅情况表”窗口
            BorrowBookList BorrowBookListFrame = new BorrowBookList();
            Dimension FrameSize = BorrowBookListFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
            //设置窗口的大小和位置
            BorrowBookListFrame.setLocation(
                    (MainFrameSize.width - FrameSize.width) / 2 + loc.x,
                    (MainFrameSize.height - FrameSize.height) / 2 + loc.y);
            BorrowBookListFrame.pack();
            //显示窗口
            BorrowBookListFrame.show();
        } else if (e.getActionCommand() == "用户列表") {
            //显示“用户列表”窗口
            UserList UserListFrame = new UserList();
            Dimension FrameSize = UserListFrame.getPreferredSize();
            Dimension MainFrameSize = getSize();
            Point loc = getLocation();
            //设置窗口的大小和位置
            UserListFrame.setLocation((MainFrameSize.width - FrameSize.width)
                    / 2 + loc.x, (MainFrameSize.height - FrameSize.height) / 2
                    + loc.y);
            UserListFrame.pack();
            //显示窗口
            UserListFrame.show();
        } else if (e.getActionCommand() == "退出") {
            this.dispose();
            //系统退出
            System.exit(0);
        }
    }
```

该方法为书籍管理系统的各个菜单项添加了事件响应函数，并设置了窗口的大小和位置。

8.8.3　为系统增加权限控制

本小节为系统增加权限控制，从而保证系统的安全性，共分为系统管理员、书籍管理员和借阅管理员 3 个角色。通过 Menu 对象的 setEnabled（true）方法，根据不同角色的权限分配设置各个菜单项的显示与否。其中的系统管理员具有全部的权限。

跟我做

在“LibraryWindow.java”中创建“setEnable”方法，编辑该方法，输入如下代码：

```
public void setEnable(String powerType) {
        if (powerType.trim().equals("系统管理员")) {
            //系统管理员具有全部的权限
                userMGRMenu.setEnabled(true);
                bookMGRMenu.setEnabled(true);
                borrowBookMenu.setEnabled(true);
                returnBookMenu.setEnabled(true);
                infoBrowseMenu.setEnabled(true);
                userListMenuItem.setEnabled(true);
        } else if (powerType.trim().equals("书籍管理员")) {
                //书籍管理员拥有书籍管理和信息查询权限
                userMGRMenu.setEnabled(false);
                bookMGRMenu.setEnabled(true);
                borrowBookMenu.setEnabled(false);
                returnBookMenu.setEnabled(false);
                infoBrowseMenu.setEnabled(true);
                userListMenuItem.setEnabled(false);

        } else if (powerType.trim().equals("借阅管理员")) {
                //借阅管理员拥有借书管理、还书管理和信息查询权限
                userMGRMenu.setEnabled(false);
                bookMGRMenu.setEnabled(false);
                borrowBookMenu.setEnabled(true);
                returnBookMenu.setEnabled(true);
                infoBrowseMenu.setEnabled(true);
                userListMenuItem.setEnabled(false);
        } else if (powerType.trim().equals("else")) {
                //其他角色没有任何权限
                userMGRMenu.setEnabled(false);
                bookMGRMenu.setEnabled(false);
                borrowBookMenu.setEnabled(false);
                returnBookMenu.setEnabled(false);
                infoBrowseMenu.setEnabled(false);
        }
}
```

8.9　用户管理

用户管理包括用户登录和用户登录信息维护两大部分，用户管理又包括添加用户、删除用户和修改用户 3 部分功能。本节实现用户管理功能。用户管理功能是典型的数据库操作，包括记录的添加、删除和修改等操作。经过本节的学习可以熟练掌握通过 Hibernate 实现对数据库的操作。

8.9.1　用户登录功能的实现

根据实现了图书管理系统的用户登录类，根据用户名和密码查询用户相应的权限。根据查询的权限结果，设置相应菜单项的显示与否。取得用户名和用户密码之后，通过如下代码段从数据库中查询相应的用户信息，如果查询的结果为空，说明该用户没有注册，将提示用户为非法用户。

```
String hql = "from UserTable where UserName='"
                        + userNameText.getText().trim() + "' and Password='"
                        + passwordStr + "'";
// 执行查询
Query userList = session.createQuery(hql);
```

跟我做

在“Library”工程的“src”文件夹中创建“library.user”包，并在其中创建“UserLogin.java”文件，在该文件中输入如下代码：

```
package library.user;

/**
 * 用户登录类，根据用户名和密码查询用户相应的权限
 *
 * @author lianhw
 *
 */
public class UserLogin extends JFrame implements ActionListener {

    LibraryWindow libraryWindow;

    JPanel panel1, panel2;

    JLabel userNameLabel, passwordLabel;

    JTextField userNameText;

    JPasswordField passwordText;

    JButton yesButton, cancelButton;

    Container container;

    ResultSet rs;

    public UserLogin(LibraryWindow mainFrame) {
        super("用户登录");
        this.libraryWindow = mainFrame;
```

```
        // “用户名”标签
        userNameLabel = new JLabel("用户名", JLabel.CENTER);
        // “密码”标签
        passwordLabel = new JLabel("密码", JLabel.CENTER);
        // 用户名输入框
        userNameText = new JTextField(10);
        // 密码输入框
        passwordText = new JPasswordField(10);
        // “确定”按钮
        yesButton = new JButton("确定");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“确定”按钮增加事件监听者
        yesButton.addActionListener(this);
        // 为“取消”按钮增加事件监听者
        cancelButton.addActionListener(this);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(2, 2));
        panel2 = new JPanel();
        container = getContentPane();
        container.setLayout(new BorderLayout());
        panel1.add(userNameLabel);
        panel1.add(userNameText);
        panel1.add(passwordLabel);
        panel1.add(passwordText);
        container.add(panel1, BorderLayout.CENTER);
        panel2.add(yesButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
        // 设置窗口大小
        setSize(300, 300);
    }

    /**
     * 动作响应方法，完成数据库的查询和权限的设置功能
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {

        if (action.getSource() == cancelButton) {
            // 如果单击“取消”按钮后执行的动作
            libraryWindow.setEnable("else");
            this.dispose();
        } else {
            // 如果单击“确定”按钮后执行的动作
            char[] password = passwordText.getPassword();
            // 将用户输入的密码由字符数组转换成字符串
            String passwordStr = new String(password);
```

```
                        // 判断用户输入的用户名是否为空
                        if (userNameText.getText().trim().equals("")) {
                            JOptionPane.showMessageDialog(null, "用户名不可为空!");
                            return;
                        }
                        // 判断用户输入的密码是否为空
                        if (passwordStr.equals("")) {
                            JOptionPane.showMessageDialog(null, "密码不可为空!");
                            return;
                        }
                        // 取得 SessionFactory
                        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
                        // 打开 session
                        Session session = sessionFactory.openSession();
                        // 创建一个事务
                        Transaction tx = session.beginTransaction();
                        // hsql 执行语句
                        String hql = "from UserTable where UserName='"
                                + userNameText.getText().trim() + "' and Password='"
                                + passwordStr + "'";
                        // 执行查询
                        Query userList = session.createQuery(hql);
                        // 将查询结果放置到一个 list 链表中
                        List list = userList.list();
                        // 标记该用户是否存在
                        boolean isExist = false;
                        // 如果 list 的长度为 0，表示该用户不存在
                        if (list.size() > 0)
                            isExist = true;
                        if (!isExist) {
                            // 提示用户名和密码不正确
                            JOptionPane.showMessageDialog(null, "用户名不存在或者密码不正确!");
                            libraryWindow.setEnable("else");
                        } else {
                            // 取得该用户的权限级别
                            UserTable user = (UserTable) list.get(0);
                            // 设置权限
                            libraryWindow.setEnable(user.getPower().trim());
                            // 事务提交
                            tx.commit();
                            // 关闭 session
                            session.close();
                            this.dispose();
                        }
                    }
                }
}
```

该类实现了用户登录功能，其运行效果如图 8-17 所示。根据用户输入的用户名和密码查询数据库中的 UserTable 表验证该用户是否是注册用户。

该类使用 Hibernate 从数据库中查询需要的信息，通过验证后根据用户不同的权限设置其相应权限。



图 8-17　用户登录窗口

8.9.2　添加用户类的实现

本小节实现增加用户类，将新用户信息保存到数据库中，从而完成用户的注册。其运行效果如图 8-18 所示。将新用户的用户名、密码和登录权限等信息提交到数据库中保存。

该类使用 Hibernate 实现了数据库的插入或更新操作，通过本类可以看出 Hibernate 操作数据库完全以面向对象的方式来实现。



图 8-18　新增用户效果图

跟我做

在“Library”工程的“library.user”包中创建“UserAdd.java”文件，在该文件中输入如下代码：

```
package library.user;


/**
 * 增加用户类，将用户信息保存到数据库中
 *
 * @author lianhw
 *
 */
public class UserAdd extends JFrame implements ActionListener {



	Container container;

	JPanel panel1, panel2;

	JLabel userNameLabel, passwordLabel, passwordConfirmLabel, loginPrivelegeLabel;

	JTextField userNameText;

	JPasswordField passwordText, passwordConfirmText;

	JComboBox loginPrivelegeComboBox;

	JButton addButton, cancelButton;
```

```
public UserAdd() {
	super("添加用户");
	container = getContentPane();
	container.setLayout(new BorderLayout());
	//“用户名”标签
	userNameLabel = new JLabel("用户名", JLabel.CENTER);
	//“密码”标签
	passwordLabel = new JLabel("密码", JLabel.CENTER);
	//“确认密码”标签
	passwordConfirmLabel = new JLabel("确认密码", JLabel.CENTER);
	//“登录权限”标签
	loginPrivelegeLabel = new JLabel("登录权限", JLabel.CENTER);
	//输入用户名的文本框
	userNameText = new JTextField(10);
	//输入密码的文本框
	passwordText = new JPasswordField(10);
	//密码确认文本框
	passwordConfirmText = new JPasswordField(10);
	//选择用户权限
	loginPrivelegeComboBox = new JComboBox();
	loginPrivelegeComboBox.addItem("系统管理员");
	loginPrivelegeComboBox.addItem("书籍管理员");
	loginPrivelegeComboBox.addItem("借阅管理员");
	//“添加”按钮
	addButton = new JButton("添加");
	//“取消”按钮
	cancelButton = new JButton("取消");
	//为“添加”按钮加入事件监听者
	addButton.addActionListener(this);
	//为“取消”按钮加入事件监听者
	cancelButton.addActionListener(this);
	panel1 = new JPanel();
	panel1.setLayout(new GridLayout(4, 2));
	panel1.add(userNameLabel);
	panel1.add(userNameText);
	panel1.add(passwordLabel);
	panel1.add(passwordText);
	panel1.add(passwordConfirmLabel);
	panel1.add(passwordConfirmText);
	panel1.add(loginPrivelegeLabel);
	panel1.add(loginPrivelegeComboBox);
	container.add(panel1, BorderLayout.CENTER);
	panel2 = new JPanel();
	panel2.add(addButton);
	panel2.add(cancelButton);
	container.add(panel2, BorderLayout.SOUTH);
	//设置窗口的大小
	setSize(300, 300);
```

```
}

/**
 * 动作响应方法，将新增的用户信息提交到数据库中
 *
 * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
 */
public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
                // 单击“取消”按钮不作任何事情
                this.dispose();
        } else if (action.getSource() == addButton) {
                // 单击“添加”按钮后将用户信息提交到数据库中

                if (userNameText.getText().trim().equals("")) {
                        // 判断用户名是否为空
                        JOptionPane.showMessageDialog(null, "用户名不能为空！");
                } else if (passwordText.getText().trim().equals("")) {
                        // 判断密码是否为空
                        JOptionPane.showMessageDialog(null, "密码不能为空！");
                } else if (!passwordText.getText().trim().equals(
                                passwordConfirmText.getText().trim())) {
                        // 判断两次输入的密码是否一致
                        JOptionPane.showMessageDialog(null, "两次输入的密码不一致！");
                } else {
                        // 取得 SessionFactory
                        SessionFactory sessionFactory = HibernateUtil
                                        .getSessionFactory();
                        // 打开 session
                        Session session = sessionFactory.openSession();
                        // 创建一个事务
                        Transaction tx = session.beginTransaction();
                        //创建 UserTable 对象
                        UserTable user = new UserTable();
                        //设置 user 对象的名字
                        user.setUserName(userNameText.getText().trim());
                        //设置 user 对象的密码
                        user.setPassword(passwordText.getText().trim());
                        //设置 usr 对象的权限
                        user.setPower(loginPrivelegeComboBox.getSelectedItem() + "");
                        //保存 user 对象
                        session.saveOrUpdate(user);

                        // 事务提交
                        tx.commit();
                        // 关闭 session
                        session.close();
```

```
                    this.dispose();

                }

            }
        }
}
```

8.9.3　修改用户信息类的实现

本小节实现了修改用户信息类，将新的用户注册信息保存到数据库中。其运行效果如图 8-19 所示。

该类通过 Hibernatyye 完全以面向对象的方式实现对数据库的更新操作。



图 8-19　修改用户信息效果图

跟我做

在“Library”工程的“library.user”包中创建“UserModify.java”文件，在该文件中输入如下代码：

```
package library.user;

/**
 * 修改用户信息类，将新的用户注册信息保存到数据库中
 *
 * @author lianhw
 *
 */
public class UserModify extends JFrame implements ActionListener {

    JPanel panel1, panel2;

    Container container;

    JLabeluserLabel,passwordLabel,newPasswordLabel,passwordConfirmLabel,loginPrivelege
Label;

    JTextField userNameText;

    JPasswordField passwordText, newPasswordText, passwordConfirmText;

    JButton updateButton, cancelButton;

    JComboBox loginPrivelegeComboBox;

    public UserModify() {
        super("更改密码");
        container = getContentPane();
```

```
container.setLayout(new BorderLayout());
// “用户名”标签
userLabel = new JLabel("用户名", JLabel.CENTER);
// “原密码”标签
passwordLabel = new JLabel("原密码", JLabel.CENTER);
// “新密码”标签
newPasswordLabel = new JLabel("新密码", JLabel.CENTER);
// “确认新密码”标签
passwordConfirmLabel = new JLabel("确认新密码", JLabel.CENTER);
//“选择权限”标签
loginPrivelegeLabel=new JLabel("登录权限", JLabel.CENTER);
// 输入用户名文本框
userNameText = new JTextField(10);
// 输入密码文本框
passwordText = new JPasswordField(10);
// 输入新密码文本框
newPasswordText = new JPasswordField(10);
// 密码确认文本框
passwordConfirmText = new JPasswordField(10);
// 选择用户权限
loginPrivelegeComboBox = new JComboBox();
loginPrivelegeComboBox.addItem("系统管理员");
loginPrivelegeComboBox.addItem("书籍管理员");
loginPrivelegeComboBox.addItem("借阅管理员");
// “更新”按钮
updateButton = new JButton("更新");
// “取消”按钮
cancelButton = new JButton("取消");
// 为“更新”按钮添加动作监听者
updateButton.addActionListener(this);
// 为“取消”按钮添加动作监听者
cancelButton.addActionListener(this);
panel1 = new JPanel();
panel1.setLayout(new GridLayout(5, 2));
panel1.add(userLabel);
panel1.add(userNameText);
panel1.add(passwordLabel);
panel1.add(passwordText);
panel1.add(newPasswordLabel);
panel1.add(newPasswordText);
panel1.add(passwordConfirmLabel);
panel1.add(passwordConfirmText);
panel1.add(loginPrivelegeLabel);
panel1.add(loginPrivelegeComboBox);
panel2 = new JPanel();
panel2.add(updateButton);
panel2.add(cancelButton);
container.add(panel1, BorderLayout.CENTER);
container.add(panel2, BorderLayout.SOUTH);
```

```
        // 设置窗口的大小
        setSize(300, 300);

    }

    /**
     * 动作响应方法，修改用户的注册信息
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
            // 单击“取消”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == updateButton) {

            // 单击“更新”按钮将新的用户信息保存到数据库中
            char[] password = passwordText.getPassword();
            // 将密码转成字符串
            String passwordStr = new String(password);
            char[] newPassword = newPasswordText.getPassword();
            String newPasswordStr = new String(newPassword);
            char[] confirmPassword = passwordConfirmText.getPassword();
            String confirmPasswordStr = new String(confirmPassword);
            // 判断输入的用户名是否为空
            if (userNameText.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "用户名不能为空！");
            } else if (passwordStr.equals("")) {
                // 判断输入的密码是否为空
                JOptionPane.showMessageDialog(null, "原密码不能为空！");
            } else if (!newPasswordStr.equals(confirmPasswordStr)) {
                // 判断输入的新密码和确认密码是否一致
                JOptionPane.showMessageDialog(null, "两次输入的新密码不一致！");
            } else {

                // 取得 SessionFactory
                SessionFactory sessionFactory = HibernateUtil
                        .getSessionFactory();
                // 打开 session
                Session session = sessionFactory.openSession();
                // 创建一个事务
                Transaction tx = session.beginTransaction();
                // 创建 UserTable 对象
                UserTable user = new UserTable();
                // 设置 user 对象的名字
                user.setUserName(userNameText.getText().trim());
                // 设置 user 对象的密码
                user.setPassword(newPasswordText.getText().trim());
                // 设置 user 对象的权限
```

```
                    user.setPower(loginPrivelegeComboBox.getSelectedItem()+"");
                    // 保存 user 对象
                    session.saveOrUpdate(user);
                    // 事务提交
                    tx.commit();
                    // 关闭 session
                    session.close();
                    this.dispose();
                }
            }
        }
}
```

8.9.4　删除用户类的实现



图 8-20　删除用户效果图

本小节实现了删除用户类，将一些用户从系统中删除。其运行效果如图 8-20 所示。

该类通过 Hibernate 实现了从数据库中删除记录的功能。

跟我做

在“Library”工程的 “library.user”包中创建“UserDelete.java”文件，在该文件中输入如下代码：

```
package library.user;

/**
 *
 * 删除用户类，将一些用户从系统中删除
 *
 * @author lianhw
 *
 */
public class UserDelete extends JFrame implements ActionListener {

    JPanel panel1, panel2;

    Container container;

    JLabel userLabel, passwordLabel;

    JTextField userText;

    JPasswordField passwordText;

    JButton yesButton, cancelButton;

    public UserDelete() {
```

```
        super("删除用户");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        //“用户名”标签
        userLabel = new JLabel("用户名", JLabel.CENTER);
        //“密码”标签
        passwordLabel = new JLabel("密码", JLabel.CENTER);
        //输入用户名文本框
        userText = new JTextField(10);
        //输入密码文本框
        passwordText = new JPasswordField(10);
        //“确定”按钮
        yesButton = new JButton("确定");
        //“取消”按钮
        cancelButton = new JButton("取消");
        //为“确定”按钮添加事件监听者
        yesButton.addActionListener(this);
        //为“取消”按钮添加事件监听者
        cancelButton.addActionListener(this);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(2, 2));
        panel1.add(userLabel);
        panel1.add(userText);
        panel1.add(passwordLabel);
        panel1.add(passwordText);
        panel2 = new JPanel();
        panel2.add(yesButton);
        panel2.add(cancelButton);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
        //设置窗口的大小
        setSize(300, 300);
    }

    /**
     * 动作响应方法，将指定用户从数据库中删除
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {

        if (action.getSource() == cancelButton) {
            // 如果单击“取消”按钮后执行的动作
            this.dispose();
        } else if (action.getSource() == yesButton) {
            // 单击“确定”按钮后将用户从数据库中删除
            char[] password = passwordText.getPassword();
            // 将用户密码转换成字符串
            String passwordStr = new String(password);
```

```
                    // 判断用户名是否为空
                    if (userText.getText().trim().equals("")) {
                        JOptionPane.showMessageDialog(null, "用户名不能为空！");
                    }
                    // 判断密码是否为空
                    else if (passwordText.equals("")) {
                        JOptionPane.showMessageDialog(null, "密码不能为空！");
                    } else {
                        // 取得 SessionFactory
                        SessionFactory sessionFactory = HibernateUtil
                                .getSessionFactory();
                        // 打开 session
                        Session session = sessionFactory.openSession();
                        // 创建一个事务
                        Transaction tx = session.beginTransaction();
                        // 创建 UserTable 对象
                        UserTable user = new UserTable();
                        // 设置 user 对象的名字
                        user.setUserName(userText.getText().trim());
                        // 设置 user 对象的密码
                        user.setPassword(passwordText.getText().trim());
                        // 删除该用户
                        session.delete(user);
                        JOptionPane.showMessageDialog(null, "删除成功!");
                        // 事务提交
                        tx.commit();
                        // 关闭 session
                        session.close();
                        this.dispose();
                    }
                }
            }
}
```

8.9.5　列举所有用户信息类的实现

本小节从数据库中查询所有注册的用户信息，并以表格的方式列出来。其运行效果如图 8-21 所示。该类利用 Hibernate 实现了数据库的查询操作。



图 8-21　用户列表一览

跟我做

在“Library”工程的“library.user”包中创建“UserList.java”文件，在该文件中输入

如下代码：

```
package library.user;

/**
 * 列出数据库中注册的所有用户信息
 *
 * @author lianhw
 *
 */
public class UserList extends JFrame {

	Container container;

	JTable table = null;

	DefaultTableModel defaultModel = null;

	public UserList() {
		super("用户列表一览！");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		// 表的两个列名
		String[] name = { "用户名", "权限" };
		String[][] data = new String[0][0];
		// 表对应的 model
		defaultModel = new DefaultTableModel(data, name);
		// 新建表格
		table = new JTable(defaultModel);
		table.setPreferredScrollableViewportSize(new Dimension(400, 80));
		JScrollPane scrollPane = new JScrollPane(table);
		container.add(scrollPane);
		// 取得 SessionFactory
		SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
		// 打开 session
		Session session = sessionFactory.openSession();
		// 创建一个事务
		Transaction tx = session.beginTransaction();
		// hsql 执行语句
		String hql = "from UserTable";
		// 执行查询
		Query userList = session.createQuery(hql);
		// 将查询结果放置到一个 list 链表中
		List list = userList.list();
		// 将链表中的数据加入到列表中
		for (int index = 0; index < list.size(); index++) {
			Vector insertRow = new Vector();
			insertRow.addElement(((UserTable) list.get(index)).getUserName());
			insertRow.addElement(((UserTable) list.get(index)).getPower());
			defaultModel.addRow(insertRow);
```

```
                }
                table.revalidate();
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();
        }
}
```

8.10　书籍管理模块

书籍管理模块实现新增书名、修改图书信息和删除图书等功能，本节实现该功能模块。

8.10.1　书籍添加类的实现

本小节实现书籍添加类，将新增加的图书信息保存到数据库中。其运行效果如图 8-22 所示。每本书的信息包括名称、出版社、作者和地址等信息。单击【添加】按钮后将添加的书籍信息通过 Hibernate 插入到数据库中。



图 8-22　添加书籍

跟我做

在“Library”工程的“src”文件夹中创建“library.book”包，并在其中创建“BookAdd.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 书籍添加类，将新增加的图书信息保存到数据库中去
 *
 * @author lianhw
 *
 */
public class BookAdd extends JFrame implements ActionListener {
                JPanel panel1, panel2;
        JLabel bookNameLabel, pressNameLabel, authorLabel, addressLabel,
                        pressDateLabel, priceLabel, bookCountLabel, commentLabel;
        JTextField bookNameText, pressNameText, authorText, addressText,
```

```
                pressDateText, priceText, bookCountText, commentText;
    Container container;
    JButton clearButton, addButton, exitButton;
    public BookAdd() {
        super("添加书籍信息");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        // “名称”标签
        bookNameLabel = new JLabel("名称", JLabel.CENTER);
        // “出版社”标签
        pressNameLabel = new JLabel("出版社", JLabel.CENTER);
        // “作者”标签
        authorLabel = new JLabel("作者", JLabel.CENTER);
        // “地址”标签
        addressLabel = new JLabel("地址", JLabel.CENTER);
        // “出版日期”标签
        pressDateLabel = new JLabel("出版日期", JLabel.CENTER);
        // “价格”标签
        priceLabel = new JLabel("价格", JLabel.CENTER);
        // “新书数目”标签
        bookCountLabel = new JLabel("新书数目", JLabel.CENTER);
        // “备注”标签
        commentLabel = new JLabel("备注", JLabel.CENTER);
        //输入书名文本框
        bookNameText = new JTextField(15);
        //输入出版社名字文本框
        pressNameText = new JTextField(15);
        //输入作者姓名文本框
        authorText = new JTextField(15);
        //输入地址文本框
        addressText = new JTextField(15);
        //输入出版日期文本框
        pressDateText = new JTextField(15);
        //输入价格文本框
        priceText = new JTextField(15);
        //输入图书数量文本框
        bookCountText = new JTextField(15);
        //输入图书备注文本框
        commentText = new JTextField(15);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(8, 2));
        panel1.add(bookNameLabel);
        panel1.add(bookNameText);
        panel1.add(pressNameLabel);
        panel1.add(pressNameText);
        panel1.add(authorLabel);
        panel1.add(authorText);
        panel1.add(addressLabel);
        panel1.add(addressText);
```

```
        panel1.add(pressDateLabel);
        panel1.add(pressDateText);
        panel1.add(priceLabel);
        panel1.add(priceText);
        panel1.add(bookCountLabel);
        panel1.add(bookCountText);
        panel1.add(commentLabel);
        panel1.add(commentText);
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 3));
        // “清空”按钮
        clearButton = new JButton("清空");
        //为“清空”按钮添加事件监听者
        clearButton.addActionListener(this);
        // “添加”按钮
        addButton = new JButton("添加");
        //为“添加”按钮添加事件监听者
        addButton.addActionListener(this);
        // “退出”按钮
        exitButton = new JButton("退出");
        //为“退出”按钮添加事件监听者
        exitButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(addButton);
        panel2.add(exitButton);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
    }
    /**
     * 动作响应方法，将新增的图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == exitButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == clearButton) {
            // 单击“清空”按钮将所有输入框清空
            bookNameText.setText("");
            pressNameText.setText("");
            authorText.setText("");
            addressText.setText("");
            pressDateText.setText("");
            priceText.setText("");
            bookCountText.setText("");
            commentText.setText("");
        } else if (action.getSource() == addButton) {
            // 判断书名是否为空
```

```
                if (bookNameText.getText().trim().equals("")) {
                    JOptionPane.showMessageDialog(null, "书名不能为空！");
                } else if (pressNameText.getText().trim().equals("")) {
                    // 判断出版社是否为空
                    JOptionPane.showMessageDialog(null, "出版社不能为空！");
                } else if (authorText.getText().trim().equals("")) {
                    // 判断作者是否为空
                    JOptionPane.showMessageDialog(null, "作者不能为空！");
                } else if (bookCountText.getText().trim().equals("")) {
                    // 判断新书数目是否为空
                    JOptionPane.showMessageDialog(null, "新书数目不能为空！");
                } else {

                    // 取得 SessionFactory
                    SessionFactory sessionFactory = HibernateUtil
                            .getSessionFactory();
                    // 打开 session
                    Session session = sessionFactory.openSession();
                    // 创建一个事务
                    Transaction tx = session.beginTransaction();
                    // 创建 UserTable 对象
                    Books book = new Books();
                    book.setBookName(bookNameText.getText().trim());
                    book.setPress(pressNameText.getText().trim());
                    book.setAuthor(authorText.getText().trim());
                    book.setAddress(addressText.getText().trim());
                    book.setPressDate(new GregorianCalendar().getTime());
                    book.setPrice(new Double(priceText.getText().trim()));
                    book.setBooksCount(new Integer(bookCountText.getText().trim()));
                    book.setCom(commentText.getText().trim());
                    session.saveOrUpdate(book);

                    // 事务提交
                    tx.commit();
                    // 关闭 session
                    session.close();

                    JOptionPane.showMessageDialog(null, "添加书籍成功！");
                    this.dispose();
                }
            }
        }
}
```

8.10.2　书籍信息修改类的实现

本小节实现图书信息修改类，将修改后的图书信息提交到数据库中。其运行效果如图 8-23 所示。该类通过 Hibernate 实现了对数据库的查询和更新操作。

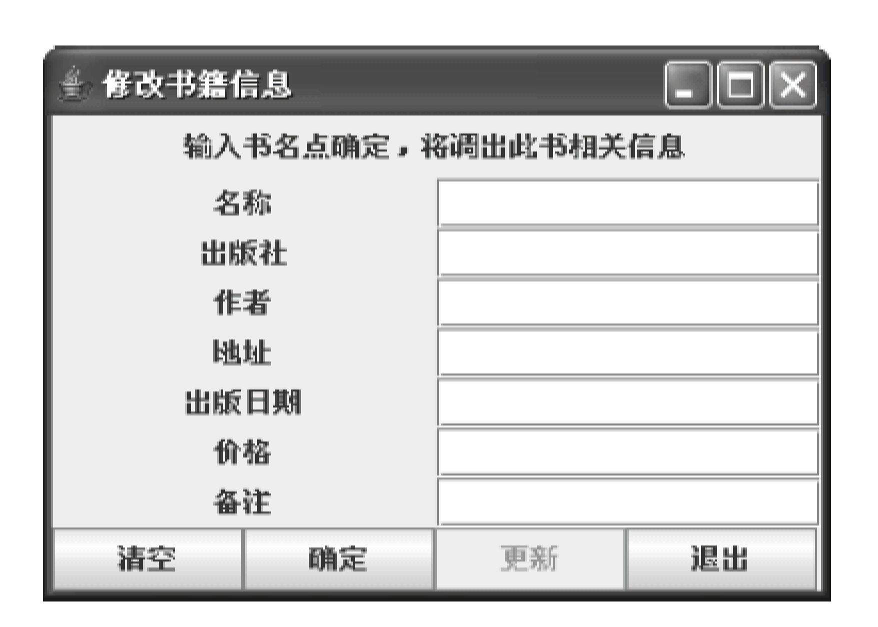


图 8-23　修改图书信息

跟我做

在“library.book”包中创建“BookModify.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 图书信息修改类，将修改后的图书信息提交到数据库中
 *
 * @author lianhw
 *
 */
public class BookModify extends JFrame implements ActionListener {

	JPanel panel1, panel2, panel3;

	JLabel tipLabel = new JLabel("输入书名点确定，将调出此书相关信息");

	JLabel bookNameLabel, pressNameLabel, authorLabel, addressLabel,
			pressDateLabel, priceLabel, commentLabel;

	JTextField bookNameText, pressNameText, authorText, addressText,
			pressDateText, priceText, commentText;

	Container container;

	JButton clearButton, yesButton, updateButton, exitButton;

	// 用来保存图书的数量
	private int count;

	/**
	 * 类的构造函数，完成界面的初始化
	 */
	public BookModify() {
		super("修改书籍信息");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		panel3 = new JPanel();
		panel3.add(tipLabel);
```

```
container.add(panel3, BorderLayout.NORTH);
// “名称”标签
bookNameLabel = new JLabel("名称", JLabel.CENTER);
// “出版社”标签
pressNameLabel = new JLabel("出版社", JLabel.CENTER);
// “作者”标签
authorLabel = new JLabel("作者", JLabel.CENTER);
// “地址”标签
addressLabel = new JLabel("地址", JLabel.CENTER);
// “出版日期”标签
pressDateLabel = new JLabel("出版日期", JLabel.CENTER);
// “价格”标签
priceLabel = new JLabel("价格", JLabel.CENTER);
// “备注”标签
commentLabel = new JLabel("备注", JLabel.CENTER);
// 书籍名称文本框
bookNameText = new JTextField(15);
// 输入出版社名称文本框
pressNameText = new JTextField(15);
// 输入作者文本框
authorText = new JTextField(15);
// 输入地址文本框
addressText = new JTextField(15);
// 输入出版日期文本框
pressDateText = new JTextField(15);
// 输入价格文本框
priceText = new JTextField(15);
// 输入备注信息文本框
commentText = new JTextField(15);
panel1 = new JPanel();
panel1.setLayout(new GridLayout(7, 2));
panel1.add(bookNameLabel);
panel1.add(bookNameText);
panel1.add(pressNameLabel);
panel1.add(pressNameText);
panel1.add(authorLabel);
panel1.add(authorText);
panel1.add(addressLabel);
panel1.add(addressText);
panel1.add(pressDateLabel);
panel1.add(pressDateText);
panel1.add(priceLabel);
panel1.add(priceText);
panel1.add(commentLabel);
panel1.add(commentText);
panel2 = new JPanel();
panel2.setLayout(new GridLayout(1, 4));
// “清空”按钮
clearButton = new JButton("清空");
```

```
        // “确定”按钮
        yesButton = new JButton("确定");
        // “更新”按钮
        updateButton = new JButton("更新");
        // “退出”按钮
        exitButton = new JButton("退出");
        panel2.add(clearButton);
        panel2.add(yesButton);
        panel2.add(updateButton);
        panel2.add(exitButton);
        // 为“清空”按钮添加监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加监听者
        yesButton.addActionListener(this);
        // 为“更新”按钮添加监听者
        updateButton.addActionListener(this);
        // 为“退出”按钮添加监听者
        exitButton.addActionListener(this);
        updateButton.setEnabled(false);
        container.add(panel1, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
    }

    /**
     * 动作响应方法，将修改后的图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == exitButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == clearButton) {
            // 单击“清空”按钮将所有文本框中的内容清空
            bookNameText.setText("");
            pressNameText.setText("");
            authorText.setText("");
            addressText.setText("");
            pressDateText.setText("");
            priceText.setText("");
            commentText.setText("");
        } else if (action.getSource() == yesButton) {
            // 单击“确定”按钮将图书信息读出
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
```

```
        // hsql 执行语句
        String hql = "from Books where bookName='"
                + bookNameText.getText().trim() + "'";
        // 执行查询
        Query bookInfoList = session.createQuery(hql);
        // 将查询结果放置到一个 list 链表中
        List list = bookInfoList.list();

        if (bookNameText.getText().trim().equals("")) {
            JOptionPane.showMessageDialog(null, "请输入书名：<*v*>");
        } else if (list.size() == 0) {
            JOptionPane.showMessageDialog(null, "此书没有在书库中...");
        } else {
            Books book = (Books) list.get(0);
            bookNameText.setText(book.getBookName());
            pressNameText.setText(book.getPress());
            authorText.setText(book.getAuthor());
            addressText.setText(book.getAddress());
            pressDateText.setText(book.getPressDate().toString());
            priceText.setText(book.getPrice().toString());
            commentText.setText(book.getCom());
            count = book.getBooksCount().intValue();
            updateButton.setEnabled(true);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();
        }

    } else if (action.getSource() == updateButton) {

        // 取得 SessionFactory
        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        // 打开 session
        Session session = sessionFactory.openSession();
        // 创建一个事务
        Transaction tx = session.beginTransaction();
        // 创建 UserTable 对象
        Books book = new Books();
        book.setBookName(bookNameText.getText().trim());
        book.setPress(pressNameText.getText().trim());
        book.setAuthor(authorText.getText().trim());
        book.setAddress(addressText.getText().trim());
        book.setPressDate(new GregorianCalendar().getTime());
        book.setPrice(new Double(priceText.getText().trim()));
        book.setCom(commentText.getText().trim());
        book.setBooksCount(new Integer(count));
        session.saveOrUpdate(book);
```

```
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();

                JOptionPane.showMessageDialog(null, "修改书籍成功！");

            }
        }
}
```

8.10.3　书籍删除类的实现



图 8-24　删除书籍

本小节实现书籍删除类，将指定名字的书籍从数据库中删除。其运行效果如图 8-24 所示。该类通过 Hibernate 实现了从数据库中删除记录的操作。

跟我做

在“library.book”包中创建“BookDelete.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 书籍删除类，将指定名字的书籍从数据库中删除
 *
 * @author lianhw
 *
 */
public class BookDelete extends JFrame implements ActionListener {
    Container container;
    JLabel tipLabel = new JLabel("请输入要删除的书名：", JLabel.CENTER);
    JTextField bookDeleteText = new JTextField(15);
    JButton yesButton, exitButton;
    JPanel panel1 = new JPanel();
    public BookDelete() {
        super("删除书籍信息");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        container.add(tipLabel, BorderLayout.NORTH);
        container.add(bookDeleteText, BorderLayout.CENTER);
        // “确定”按钮
        yesButton = new JButton("确定");
        // “退出”按钮
        exitButton = new JButton("退出");
        // 为“确定”按钮添加事件监听者
        yesButton.addActionListener(this);
        // 为“退出”按钮添加事件监听者
        exitButton.addActionListener(this);
        panel1.add(yesButton);
```

```
            panel1.add(exitButton);
            container.add(panel1, BorderLayout.SOUTH);
        }

        /**
         * 动作响应方法，将修改后的图书信息提交到数据库中
         *
         * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
         */
        public void actionPerformed(ActionEvent action) {
            if (action.getSource() == exitButton) {
                // 单击“退出”按钮不作任何事情
                this.dispose();
            } else if (action.getSource() == yesButton) {
                // 单击“确定”按钮将图书从数据库中删除

                // 取得 SessionFactory
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
                // 打开 session
                Session session = sessionFactory.openSession();
                // 创建一个事务
                Transaction tx = session.beginTransaction();
                Books book = new Books();
                book.setBookName(bookDeleteText.getText().trim());
                session.delete(book);
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();

                JOptionPane.showMessageDialog(null, "删除书籍成功！");
                this.dispose();
            }
        }
}
```

8.10.4　图书信息一览类的实现

本小节实现图书信息一览类。从数据库中查询到所有的图书信息，并以表格的形式显示。其运行效果如图 8-25 所示。该类通过 Hibernate 实现了从数据库中查询记录的操作。



图 8-25　书籍信息一览

跟我做

在“library.book”包中创建“BookList.java”文件，在该文件中输入如下代码：

```
package library.book;
/**
 * 图书信息一览类
 *
 * @author lianhw
 *
 */
public class BookList extends JFrame implements ActionListener {

	Container container;

	JPanel panel1, panel2, panel3;

	JLabel bookNameLabel, authorLabel, pressLabel;

	JTextField bookNameText, authorText, pressText;

	JButton searchButton, exitButton;

	JTable table = null;

	DefaultTableModel defaultModel = null;

	public BookList() {
		super("书籍信息一览！");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		//“名称”标签
		bookNameLabel = new JLabel("名称", JLabel.CENTER);
		//“作者”标签
		authorLabel = new JLabel("作者", JLabel.CENTER);
		//“出版社”标签
		pressLabel = new JLabel("出版社", JLabel.CENTER);
		//输入书名文本框
		bookNameText = new JTextField(15);
		//输入作者姓名文本框
		authorText = new JTextField(15);
		//输入出版社社名文本框
		pressText = new JTextField(15);
		//“查询”按钮
		searchButton = new JButton("查询");
		//为“查询”按钮增加事件监听者
		searchButton.addActionListener(this);
		//“退出”按钮
		exitButton = new JButton("退出");
```

```
        //为“退出”按钮添加事件监听者
        exitButton.addActionListener(this);
        panel1 = new JPanel();
        panel3 = new JPanel();
        panel1.add(bookNameLabel);
        panel1.add(bookNameText);
        panel1.add(authorLabel);
        panel1.add(authorText);
        panel3.add(pressLabel);
        panel3.add(pressText);
        panel3.add(searchButton);
        panel3.add(exitButton);
        //表格的列名
        String[] name = { "书名", "出版社", "作者", "地址", "出版日期", "定价", "评论" };
        String[][] data = new String[0][0];
        defaultModel = new DefaultTableModel(data, name);
        table = new JTable(defaultModel);
        table.setPreferredScrollableViewportSize(new Dimension(400, 80));
        JScrollPane s = new JScrollPane(table);
        panel2 = new JPanel();
        panel2.add(s);
        container.add(panel1, BorderLayout.NORTH);
        container.add(panel3, BorderLayout.CENTER);
        container.add(panel2, BorderLayout.SOUTH);
    }

    /**
     * 动作响应方法，从数据库中查询所有图书信息
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == searchButton) {
            String hql = " from Books";
            String strSql = null;
            if (bookNameText.getText().trim().equals("")
                    && authorText.getText().trim().equals("")
                    && pressText.getText().trim().equals("")) {
                // 如果没有查询条件
                strSql = hql;
            } else if (bookNameText.getText().trim().equals("")
                    && authorText.getText().trim().equals("")) {
                // 按照出版社查询书籍
                strSql = hql + " where press='" + pressText.getText().trim()
                        + "'";
            } else if (bookNameText.getText().trim().equals("")
                    && pressText.getText().trim().equals("")) {
                // 按照作者姓名查询书籍
                strSql = hql + " where author='" + authorText.getText().trim()
```

```
                    + "'";
} else if (authorText.getText().trim().equals("")
                && pressText.getText().trim().equals("")) {
        // 按照书名查询书籍
        strSql = hql + " where bookName='"
                    + bookNameText.getText().trim() + "'";
} else if (bookNameText.getText().trim().equals("")) {
        // 按照作者和出版社两个条件来查询书籍
        strSql = hql + " where author='" + authorText.getText().trim()
                    + "'and press='" + pressText.getText().trim() + "'";
} else if (authorText.getText().trim().equals("")) {
        // 按照书名和出版社两个条件来查询书籍
        strSql = hql + " where bookName='"
                    + bookNameText.getText().trim() + "'and press='"
                    + pressText.getText().trim() + "'";
} else if (pressText.getText().trim().equals("")) {
        // 按照书名和作者两个条件来查询书籍
        strSql = hql + " where bookname='"
                    + bookNameText.getText().trim() + "'and author='"
                    + authorText.getText().trim() + "'";
} else {
        // 按照书名、作者和出版社 3 个条件来查询书籍
        strSql = hql + " where bookname='"
                    + bookNameText.getText().trim() + "'and author='"
                    + authorText.getText().trim() + "'and press='"
                    + pressText.getText().trim() + "'";
}

// 首先要删除 table 中的数据
int rowCount = defaultModel.getRowCount() - 1;// 取得 table 中的数据行；
int j = rowCount;
for (int i = 0; i <= rowCount; i++) {
        defaultModel.removeRow(j);// 删除 rowCount 行的数据；
        defaultModel.setRowCount(j);// 重新设置行数；
        j = j - 1;
}
// 取得 SessionFactory
SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();

// 执行查询
Query userList = session.createQuery(strSql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
for (int index = 0; index < list.size(); index++) {
        Vector data = new Vector();
```

```
                    Books book = (Books) list.get(index);
                    data.addElement(book.getBookName());
                    data.addElement(book.getPress());
                    data.addElement(book.getAuthor());
                    data.addElement(book.getAddress());
                    data.addElement(book.getPressDate());
                    data.addElement(book.getPrice());
                    data.addElement(book.getCom());
                    defaultModel.addRow(data);
                }
                table.revalidate();
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();

            } else if (action.getSource() == exitButton) {
                this.dispose();
            }
        }
}
```

8.11　借书管理模块

借书管理实现了对出借图书的信息维护。包括书籍出借和出借信息修改两部分功能。本节实现该功能模块。

8.11.1　借阅图书类的实现

本小节实现借阅图书类，将借阅的图书信息提交到数据库中。其运行效果如图 8-26 所示。该类通过 Hibernate 实现了对数据库的插入操作。



图 8-26　书籍出借窗口

跟我做

在“library.book”包中创建“BorrowBook.java”文件，在该文件中输入如下代码：

```
package library.book;
```

```
/**
 * 借阅图书类，将借阅的图书信息提交到数据库中
 *
 * @author lianhw
 *
 */
public class BorrowBook extends JFrame implements ActionListener {

	JPanel panel1, panel2;

	Container container;

	JLabel borrowedBookStudentLabel, borrowedBookNameLabel, borrowedDateLabel,
			returnDateLabel, borrowedCommentLabel;

	JTextField borrowedBookStudentText, borrowedDateText, returnDateText,
			borrowedCommentText;

	JButton clearButton, yesButton, cancelButton;

	JComboBox bookNameComboBox = new JComboBox();

	/**
	 * 类的构造函数，完成界面的初始化
	 */
	public BorrowBook() {
		super("书籍出借");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		// “借阅者姓名”标签
		borrowedBookStudentLabel = new JLabel("借阅者姓名", JLabel.CENTER);
		// “书名”标签
		borrowedBookNameLabel = new JLabel("书名", JLabel.CENTER);
		// “借阅日期”标签
		borrowedDateLabel = new JLabel("借阅日期", JLabel.CENTER);
		// “归还日期”标签
		returnDateLabel = new JLabel("归还日期", JLabel.CENTER);
		// “备注”标签
		borrowedCommentLabel = new JLabel("备注", JLabel.CENTER);
		// 输入借阅者姓名文本框
		borrowedBookStudentText = new JTextField(15);
		// 输入借阅日期文本框
		borrowedDateText = new JTextField(15);
		// 输入归还日期文本框
		returnDateText = new JTextField(15);
		// 输入备注信息文本框
		borrowedCommentText = new JTextField(15);
		// 取得 SessionFactory
		SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
```

```
// 打开 session
Session session = sessionFactory.openSession();
// 创建一个事务
Transaction tx = session.beginTransaction();
// hsql 执行语句
String hql = "from Books where books_count>borrowed_count";
// 执行查询
Query userList = session.createQuery(hql);
// 将查询结果放置到一个 list 链表中
List list = userList.list();
// 将查询到所有符合条件的图书加入到出借列表中
for (int index = 0; index < list.size(); index++) {
    bookNameComboBox.addItem(((Books) list.get(index)).getBookName());
}
// 事务提交
tx.commit();
// 关闭 session
session.close();

panel1 = new JPanel();
panel1.setLayout(new GridLayout(5, 2));
panel1.add(borrowedBookStudentLabel);
panel1.add(borrowedBookStudentText);
panel1.add(borrowedBookNameLabel);
panel1.add(bookNameComboBox);
panel1.add(borrowedDateLabel);
panel1.add(borrowedDateText);
panel1.add(returnDateLabel);
panel1.add(returnDateText);
panel1.add(borrowedCommentLabel);
panel1.add(borrowedCommentText);
container.add(panel1, BorderLayout.CENTER);
panel2 = new JPanel();
panel2.setLayout(new GridLayout(1, 3));
// “清空”按钮
clearButton = new JButton("清空");
// “确定”按钮
yesButton = new JButton("确定");
// “取消”按钮
cancelButton = new JButton("取消");
// 为“清空”按钮添加信息监听者
clearButton.addActionListener(this);
// 为“确定”按钮添加信息监听者
yesButton.addActionListener(this);
// 为“取消”按钮添加信息监听者
cancelButton.addActionListener(this);
panel2.add(clearButton);
panel2.add(yesButton);
panel2.add(cancelButton);
```

```
        container.add(panel2, BorderLayout.SOUTH);

    }

    /**
     * 动作响应方法，将出借的图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == cancelButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == clearButton) {
            // 单击“清空”按钮将文本框中的信息清空
            borrowedBookStudentText.setText("");
            borrowedDateText.setText("");
            borrowedCommentText.setText("");

        } else if (action.getSource() == yesButton) {
            // 验证输入的借阅者姓名是否为空
            if (borrowedBookStudentText.getText().trim().equals("")) {
                JOptionPane.showMessageDialog(null, "请输入借阅者的姓名…");
            } else if (bookNameComboBox.getSelectedItem().equals("")) {
                // 验证现在是否有书可以借阅
                JOptionPane.showMessageDialog(null, "对不起，现在书库里没有书，\n 你现
在不能借书!");
            } else {

                // 取得 SessionFactory
                SessionFactory sessionFactory = HibernateUtil
                        .getSessionFactory();
                // 打开 session
                Session session = sessionFactory.openSession();
                // 创建一个事务
                Transaction tx = session.beginTransaction();
                // 创建 UserTable 对象
                BookBrowse browse = new BookBrowse();
                browse.setBookName(bookNameComboBox.getSelectedItem() + "");
                browse.setStudentName(borrowedBookStudentText.getText().trim());
                browse.setBorrowDate(new GregorianCalendar().getTime());
                browse.setCom(borrowedCommentText.getText().trim());
                browse.setReturnDate(new GregorianCalendar().getTime());
                session.saveOrUpdate(browse);

                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();
```

```
                JOptionPane.showMessageDialog(null, "借阅书籍成功！");
                this.dispose();

            }
        }
    }
}
```

8.11.2　修改出借图书信息类的实现

本小节实现了修改出借图书信息类，将修改后的图书信息提交到数据库中。其运行效果如图 8-27 所示。该类通过 Hibernate 实现了从数据库中查询记录和更新记录的操作。

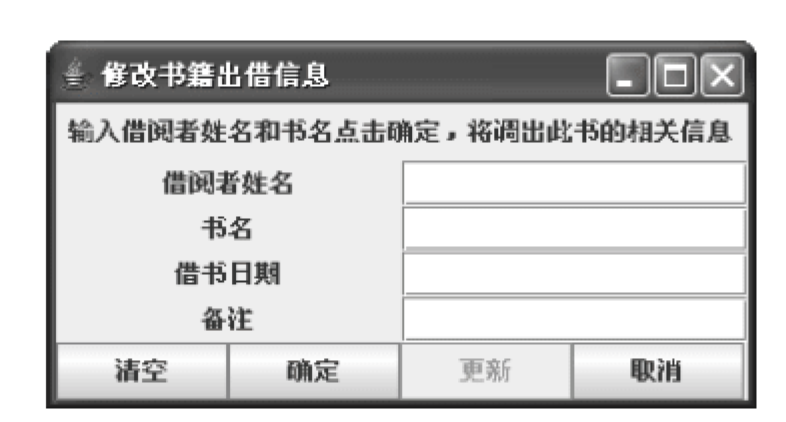


图 8-27　修改书籍出借信息

跟我做

在“Library”工程的“src”文件夹中创建“library.info”包，并在其中创建“BorrowInfo.java”文件，在该文件中输入如下代码：

```
package library.info;

/**
 * 修改出借图书信息类，将修改后的图书信息提交到数据库中
 *
 * @author lianhw
 *
 */
public class BorrowInfo extends JFrame implements ActionListener {

    JPanel panel1, panel2, panel3;

    Container container;

    JLabel tipLabel = new JLabel("输入借阅者姓名和书名点击确定，将调出此书的相关信息");

    JLabel borrowedBookStudentLabel, borrowedBookNameLabel, borrowedDateLabel,
            borrowedCommentLabel;

    JTextField borrowedBookStudentText, borrowedBookNameText, borrowedDateText,
            borrowedCommentText;
```

```
    JButton clearButton, yesButton, updateButton, cancelButton;

    public BorrowInfo() {
        super("修改书籍出借信息");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        panel3 = new JPanel();
        panel3.add(tipLabel);
        container.add(panel3, BorderLayout.NORTH);
        // “借阅者姓名”标签
        borrowedBookStudentLabel = new JLabel("借阅者姓名", JLabel.CENTER);
        // “书名”标签
        borrowedBookNameLabel = new JLabel("书名", JLabel.CENTER);
        // “借书日期”标签
        borrowedDateLabel = new JLabel("借书日期", JLabel.CENTER);
        // “备注”标签
        borrowedCommentLabel = new JLabel("备注", JLabel.CENTER);
        // 输入借阅者姓名的文本框
        borrowedBookStudentText = new JTextField(15);
        // 输入书名文本框
        borrowedBookNameText = new JTextField(15);
        // 输入出借日期文本框
        borrowedDateText = new JTextField(15);
        // 输入备注信息文本框
        borrowedCommentText = new JTextField(15);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(4, 2));
        panel1.add(borrowedBookStudentLabel);
        panel1.add(borrowedBookStudentText);
        panel1.add(borrowedBookNameLabel);
        panel1.add(borrowedBookNameText);
        panel1.add(borrowedDateLabel);
        panel1.add(borrowedDateText);
        panel1.add(borrowedCommentLabel);
        panel1.add(borrowedCommentText);
        container.add(panel1, BorderLayout.CENTER);
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 4));
        // “清空”按钮
        clearButton = new JButton("清空");
        // “确定”按钮
        yesButton = new JButton("确定");
        // “更新”按钮
        updateButton = new JButton("更新");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“清空”按钮添加监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加监听者
```

```
        yesButton.addActionListener(this);
        // 为“更新”按钮添加监听者
        updateButton.addActionListener(this);
        updateButton.setEnabled(false);
        // 为“取消”按钮添加监听者
        cancelButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(yesButton);
        panel2.add(updateButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }

    /**
     * 动作响应方法，将修改后的出借图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        // 单击“清空”按钮，清空所有的文本框
        if (action.getSource() == clearButton) {
            borrowedBookStudentText.setText("");
            borrowedBookNameText.setText("");
            borrowedDateText.setText("");
            borrowedCommentText.setText("");
        } else if (action.getSource() == cancelButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == yesButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // hsql 执行语句
            String hql = "from BookBrowse where studentName='"
                    + borrowedBookStudentText.getText().trim()
                    + "'and bookName='" + borrowedBookNameText.getText().trim()
                    + "'";
            // 执行查询
            Query userList = session.createQuery(hql);
            // 将查询结果放置到一个 list 链表中
            List list = userList.list();
            BookBrowse browse = (BookBrowse) list.get(0);
            borrowedBookStudentText.setText(browse.getStudentName());
            borrowedBookNameText.setText(browse.getBookName());
            borrowedDateText.setText(browse.getBorrowDate().toString());
            borrowedCommentText.setText(browse.getCom());
```

```
                updateButton.setEnabled(true);
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();

            } else if (action.getSource() == updateButton) {
                // 取得 SessionFactory
                SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
                // 打开 session
                Session session = sessionFactory.openSession();
                // 创建一个事务
                Transaction tx = session.beginTransaction();
                BookBrowse browse2 = new BookBrowse();
                browse2.setBookName(borrowedBookNameText.getText().trim());
                browse2.setStudentName(borrowedBookStudentText.getText().trim());
                browse2.setBorrowDate(new GregorianCalendar().getTime());
                browse2.setCom(borrowedCommentText.getText().trim());
                browse2.setReturnDate(new GregorianCalendar().getTime());
                session.saveOrUpdate(browse2);
                JOptionPane.showMessageDialog(null, "修改书籍成功！ ");
                // 事务提交
                tx.commit();
                // 关闭 session
                session.close();
                this.dispose();
            }
        }
}
```

8.12　还书管理模块

还书管理模块实现了对还书信息的管理。包括书籍还入和书籍还入信息修改两部分功能。本节实现该功能模块。

8.12.1　还书类的实现

本小节实现所有有关还书的功能。该类实现了将还书信息通过 Hibernate 保存到数据库中，其运行效果如图 8-28 所示。



图 8-28　书籍还入

跟我做

在“library.info”包中创建“ReturnedBook.java”文件，在该文件中输入如下代码：

```
package library.book;

/**
 * 还书类
 *
 * @author lianhw
 *
 */
public class ReturnBook extends JFrame implements ActionListener {

	JPanel panel1, panel2;
	Container container;
	JLabel returnedBookStudentLabel, returnedBookNameLabel, returnedDateLabel,
			returnedCommentLabel;
	JTextField returnedBookStudentText, returnedDateText, returnedCommentText;
	JButton clearButton, yseButton, cancelButton;
	JComboBox bookNameComboBox = new JComboBox();
	// 书名和 BookBrowse 类的映射
	private Map bookName2BookBrowse = new HashMap();
	public ReturnBook() {
		super("书籍还入");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		// “还书者姓名”标签
		returnedBookStudentLabel = new JLabel("还书者姓名", JLabel.CENTER);
		// “书名”标签
		returnedBookNameLabel = new JLabel("书名", JLabel.CENTER);
		// “日期”标签
		returnedDateLabel = new JLabel("日期", JLabel.CENTER);
		// “备注”标签
		returnedCommentLabel = new JLabel("备注", JLabel.CENTER);
		// 输入归还者姓名文本框
		returnedBookStudentText = new JTextField(15);
		// 输入归还日期文本框
		returnedDateText = new JTextField(15);
		// 输入备注文本框
		returnedCommentText = new JTextField(15);
		// 取得 SessionFactory
		SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
		// 打开 session
		Session session = sessionFactory.openSession();
		// 创建一个事务
		Transaction tx = session.beginTransaction();
		// hsql 执行语句
		String hql = "from BookBrowse where is_returned='否'";
```

```
        // 执行查询
        Query userList = session.createQuery(hql);
        // 将查询结果放置到一个 list 链表中
        List list = userList.list();

        for (int index = 0; index < list.size(); index++) {
            BookBrowse bb = (BookBrowse) list.get(index);
            bookNameComboBox.addItem(bb.getBookName());
            bookName2BookBrowse.put(bb.getBookName(), bb);
        }
        // 事务提交
        tx.commit();
        // 关闭 session
        session.close();
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(4, 2));
        panel1.add(returnedBookStudentLabel);
        panel1.add(returnedBookStudentText);
        panel1.add(returnedBookNameLabel);
        panel1.add(bookNameComboBox);
        panel1.add(returnedDateLabel);
        panel1.add(returnedDateText);
        panel1.add(returnedCommentLabel);
        panel1.add(returnedCommentText);
        container.add(panel1, BorderLayout.CENTER);
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 3));
        // “清空”按钮
        clearButton = new JButton("清空");
        // “确定”按钮
        yseButton = new JButton("确定");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“清空”按钮添加事件监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加事件监听者
        yseButton.addActionListener(this);
        // 为“取消”按钮添加事件监听者
        cancelButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(yseButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }

    /**
     * 动作响应方法，将修改后的出借图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
```

```
         */
        public void actionPerformed(ActionEvent action) {
                if (action.getSource() == cancelButton) {
                        // 单击“退出”按钮不作任何事情
                        this.dispose();
                } else if (action.getSource() == clearButton) {
                        // 单击“清空”按钮，清空所有的文本框
                        returnedBookStudentText.setText("");
                        returnedDateText.setText("");
                        returnedCommentText.setText("");

                } else if (action.getSource() == yseButton) {
                        // 判断还书者的姓名是否为空
                        if (returnedBookStudentText.getText().trim().equals("")) {
                                JOptionPane.showMessageDialog(null, "请输入还书者的姓名…");
                        } else if (bookNameComboBox.getSelectedItem().equals("")) {
                                // 判断有没有出借的书
                                JOptionPane.showMessageDialog(null, "图书馆没有出借过书！");
                        } else {
                                // 取得 SessionFactory
                                SessionFactory sessionFactory = HibernateUtil
                                                .getSessionFactory();
                                // 打开 session
                                Session session = sessionFactory.openSession();
                                // 创建一个事务
                                Transaction tx = session.beginTransaction();
                                String bookName = bookNameComboBox.getSelectedItem().toString();
                                BookBrowse bookBrowse = (BookBrowse) bookName2BookBrowse
                                                .get(bookName);
                                bookBrowse.setIsReturned("是");
                                bookBrowse.setCom(returnedCommentText.getText().trim());
                                session.saveOrUpdate(bookBrowse);

                                JOptionPane.showMessageDialog(null, "还书成功！");
                                // 事务提交
                                tx.commit();
                                // 关闭 session
                                session.close();
                                this.dispose();
                        }
                }
        }
}
```

8.12.2　修改还书信息类的实现

本小节实现修改还书信息类。该类通过 Hibernate 将修改后的还书信息保存到数据库中，其运行效果如图 8-29 所示。

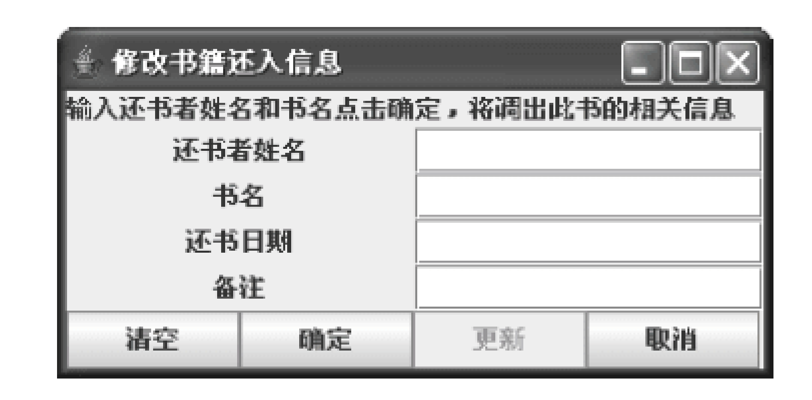


图 8-29　修改书籍还入信息

跟我做

在“library.info”包中创建 ReturnInfo.java 类，在该文件中输入如下内容：

```
package library.info;
/**
 * 修改还书信息类
 *
 * @author lianhw
 *
 */
public class ReturnInfo extends JFrame implements ActionListener {

	JPanel panel1, panel2;
	Container container;
	JLabel tipLabel = new JLabel("输入还书者姓名和书名点击确定，将调出此书的相关信息");
	JLabel returnedBookStudentLabel, returnedBookNameLabel, returnedDateLabel,
			returnedCommentLabel;
	JTextField returnedBookStudentText, returnedBookNameText, returnedDateText,
			returnedCommentText;
	JButton clearButton, yesButton, updateButton, cancelButton;

	public ReturnInfo() {
		super("修改书籍还入信息");
		container = getContentPane();
		container.setLayout(new BorderLayout());
		container.add(tipLabel, BorderLayout.NORTH);
		// “还书者姓名”标签
		returnedBookStudentLabel = new JLabel("还书者姓名", JLabel.CENTER);
		// “书名”标签
		returnedBookNameLabel = new JLabel("书名", JLabel.CENTER);
		// “还书日期”标签
		returnedDateLabel = new JLabel("还书日期", JLabel.CENTER);
		// “备注”标签
		returnedCommentLabel = new JLabel("备注", JLabel.CENTER);
		// 输入还书学生姓名文本框
		returnedBookStudentText = new JTextField(15);
		// 输入还书名称文本框
		returnedBookNameText = new JTextField(15);
		// 输入还书日期文本框
		returnedDateText = new JTextField(15);
		// 输入还书备注文本框
```

```
        returnedCommentText = new JTextField(15);
        panel1 = new JPanel();
        panel1.setLayout(new GridLayout(4, 2));
        panel1.add(returnedBookStudentLabel);
        panel1.add(returnedBookStudentText);
        panel1.add(returnedBookNameLabel);
        panel1.add(returnedBookNameText);
        panel1.add(returnedDateLabel);
        panel1.add(returnedDateText);
        panel1.add(returnedCommentLabel);
        panel1.add(returnedCommentText);
        container.add(panel1, BorderLayout.CENTER);
        panel2 = new JPanel();
        panel2.setLayout(new GridLayout(1, 4));
        // “清空”按钮
        clearButton = new JButton("清空");
        // “确定”按钮
        yesButton = new JButton("确定");
        // “更新”按钮
        updateButton = new JButton("更新");
        // “取消”按钮
        cancelButton = new JButton("取消");
        // 为“清空”按钮添加监听者
        clearButton.addActionListener(this);
        // 为“确定”按钮添加监听者
        yesButton.addActionListener(this);
        // 为“更新”按钮添加监听者
        updateButton.addActionListener(this);
        updateButton.setEnabled(false);
        // 为“取消”按钮添加监听者
        cancelButton.addActionListener(this);
        panel2.add(clearButton);
        panel2.add(yesButton);
        panel2.add(updateButton);
        panel2.add(cancelButton);
        container.add(panel2, BorderLayout.SOUTH);
    }

    /**
     * 动作响应方法，将修改后的出借图书信息提交到数据库中
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == clearButton) {
            // 单击“清空”按钮，清空所有的文本框
            returnedBookStudentText.setText("");
            returnedBookNameText.setText("");
            returnedDateText.setText("");
```

```
            returnedCommentText.setText("");
        } else if (action.getSource() == cancelButton) {
            // 单击“退出”按钮不作任何事情
            this.dispose();
        } else if (action.getSource() == yesButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            // hsql 执行语句
            String hql = "from BookBrowse where studentName='"
                    + returnedBookStudentText.getText().trim()
                    + "'and bookName='" + returnedBookNameText.getText().trim()
                    + "'";
            // 执行查询
            Query userList = session.createQuery(hql);
            // 将查询结果放置到一个 list 链表中
            List list = userList.list();
            BookBrowse browse = (BookBrowse) list.get(0);
            returnedBookStudentText.setText(browse.getStudentName());
            returnedBookNameText.setText(browse.getBookName());
            returnedDateText.setText(browse.getBorrowDate().toString());
            returnedCommentText.setText(browse.getCom());
            updateButton.setEnabled(true);
            // 事务提交
            tx.commit();
            // 关闭 session
            session.close();

        } else if (action.getSource() == updateButton) {
            // 取得 SessionFactory
            SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
            // 打开 session
            Session session = sessionFactory.openSession();
            // 创建一个事务
            Transaction tx = session.beginTransaction();
            BookBrowse browse2 = new BookBrowse();
            browse2.setBookName(returnedBookNameText.getText().trim());
            browse2.setStudentName(returnedBookStudentText.getText().trim());
            browse2.setBorrowDate(new GregorianCalendar().getTime());
            browse2.setCom(returnedCommentText.getText().trim());
            browse2.setReturnDate(new GregorianCalendar().getTime());
            session.saveOrUpdate(browse2);
            JOptionPane.showMessageDialog(null, "修改书籍成功！ ");
            // 事务提交
            tx.commit();
            // 关闭 session
```

```
                session.close();
                this.dispose();
            }
        }
}
```

8.12.3　借阅图书一览类的实现

本小节实现借阅图书一览。该类通过 Hibernate 实现了从数据库中查询所有相关记录的操作，其运行效果如图 8-30 所示。



图 8-30　书籍借阅一览

跟我做

在“library.book”包中创建 BorrowBookList.java 类，在该类中输入如下内容：

```
package library.book;
public class BorrowBookList extends JFrame implements ActionListener {
    Container container;
    JPanel panel1, panel2;
    JLabel bookNameLabel, studentNameLabel;
    JTextField bookNameText, studentNameText;
    JButton searchButton, exitButton;
    JTable table = null;
    DefaultTableModel defaultModel = null;
    public BorrowBookList() {
        super("书籍借阅一览！");
        container = getContentPane();
        container.setLayout(new BorderLayout());
        //“书名”标签
        bookNameLabel = new JLabel("书名", JLabel.CENTER);
        //“借阅者”标签
        studentNameLabel = new JLabel("借阅者", JLabel.CENTER);
        //输入“书名”文本框
        bookNameText = new JTextField(15);
        //输入“学生姓名”文本框
        studentNameText = new JTextField(15);
        //“查询”按钮
        searchButton = new JButton("查询");
        //“退出”按钮
        exitButton = new JButton("退出");
```

```
        //为“查询”按钮增加监听者
        searchButton.addActionListener(this);
        //为“退出”按钮增加监听者
        exitButton.addActionListener(this);
        Box box1 = Box.createHorizontalBox();
        box1.add(studentNameLabel);
        box1.add(studentNameText);
        box1.add(searchButton);
        Box box2 = Box.createHorizontalBox();
        box2.add(bookNameLabel);
        box2.add(bookNameText);
        box2.add(exitButton);
        Box boxH = Box.createVerticalBox();
        boxH.add(box1);
        boxH.add(box2);
        boxH.add(Box.createVerticalGlue());
        panel1 = new JPanel();
        panel1.add(boxH);
        panel2 = new JPanel();
        String[] name = { "借阅者", "书名", "借阅日期", "还入日期", "备注" };
        String[][] data = new String[0][0];
        defaultModel = new DefaultTableModel(data, name);
        table = new JTable(defaultModel);
        table.setPreferredScrollableViewportSize(new Dimension(400, 80));
        JScrollPane s = new JScrollPane(table);
        panel2.add(s);
        container.add(panel1, BorderLayout.NORTH);
        container.add(panel2, BorderLayout.SOUTH);
    }

    /**
     * 动作响应方法，查询所有图书的出借情况
     *
     * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent action) {
        if (action.getSource() == exitButton) {
            // 单击“退出”按钮，不作任何事情
            this.dispose();
        } else if (action.getSource() == searchButton) {
            String strSQL = " from BookBrowse";
            String strSql = null;
            if (studentNameText.getText().trim().equals("")
                    && bookNameText.getText().trim().equals("")) {
                // 无条件查询
                strSql = strSQL;
            } else if (studentNameText.getText().trim().equals("")) {
                // 按书名查询
                strSql = strSQL + " where bookName='"
```

```
                    + bookNameText.getText().trim() + "'";
        } else if (bookNameText.getText().trim().equals("")) {
            // 按学生姓名查询
            strSql = strSQL + " where studentName='"
                    + studentNameText.getText().trim() + "'";
        } else {
            // 按学生姓名和书名查询
            strSql = strSQL + " where studentName='"
                    + studentNameText.getText().trim() + "'and bookName='"
                    + bookNameText.getText().trim() + "'";
        }

        // 首先要删除 table 中的数据：
        int rowCount = defaultModel.getRowCount() - 1;// 取得 table 中的数据行；
        int j = rowCount;
        for (int i = 0; i <= rowCount; i++) {
            defaultModel.removeRow(j);// 删除 rowCount 行的数据；
            defaultModel.setRowCount(j);// 重新设置行数；
            j = j - 1;
        }
        // 取得 SessionFactory
        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        // 打开 session
        Session session = sessionFactory.openSession();
        // 创建一个事务
        Transaction tx = session.beginTransaction();

        // 执行查询
        Query userList = session.createQuery(strSql);
        // 将查询结果放置到一个 list 链表中
        List list = userList.list();
        for (int index = 0; index < list.size(); index++) {
            Vector data = new Vector();
            BookBrowse book = (BookBrowse) list.get(index);
            data.addElement(book.getStudentName());
            data.addElement(book.getBookName());
            data.addElement(book.getBorrowDate());
            data.addElement(book.getReturnDate());
            data.addElement(book.getCom());
            defaultModel.addRow(data);
        }
        table.revalidate();
        // 事务提交
        tx.commit();
        // 关闭 session
        session.close();
    }
  }
}
```

第 9 章　JUnit 开发实例——图书管理系统的单元测试

JUnit 是一个回归测试框架（regression testing framework）。JUnit 测试是程序员测试，即所谓白盒测试，因为程序员知道被测试的软件如何（how）完成功能和完成什么样（what）的功能。JUnit 是一套框架，继承 TestCase 类，即可用 Junit 进行自动测试。JUnit 相对独立于所编写的代码，测试代码的编写甚至可以先于实现代码的编写。

本章介绍在 Eclipse 中进行 JUnit 单元测试的基本知识。首先通过一个最简单的 HelloWorld 类的测试用例的开发来说明 JUnit 框架的基本原理，然后通过对图书管理系统进行单元测试来说明 JUnit 的具体应用。

9.1　Eclipse 与 JUnit 的集成

EclipseJDT 工具包括一个将 JUnit 集成到 JavaIDE 中的插件。JUnit 插件允许定义代码的回归测试并从 JavaIDE 中运行它们。

跟我做

（1）单击 Eclipse 的【窗口】菜单，选择【显示视图】|【其他】命令，打开【显示视图】对话框，如图 9-1 所示。

（2）单击【确定】按钮，打开 JUnit 视图，如图 9-2 所示。



图 9-1　【显示视图】对话框



图 9-2　JUnit 视图

（3）单击 Eclipse 的【窗口】菜单，选择【首选项】命令，打开如图 9-3 所示的【首选项】窗口。



图 9-3　JUnit 的首选项

此页面允许为 JUnit 配置堆栈跟踪过滤器模式。在 JUnit 视图中将使用这些模式来控制哪些条目在堆栈跟踪中可视，如表 9-1 所示。

表 9-1　JUnit首选项操作列表

操　作	描　述
添加过滤器	允许添加定制过滤器模式。此操作将插入可以编辑的空模式
添加类	允许添加要过滤的类。此操作打开类型对话框，用户可以使用该对话框来选择在堆栈跟踪中要过滤的类型
添加包	允许添加要过滤的包。此操作打开一个包选择对话框，用户可以使用该对话框来选择在堆栈跟踪中要过滤的包
除去	除去当前选择的堆栈跟踪过滤器模式
全部启用	启用所有堆栈跟踪过滤器模式
全部禁用	禁用所有堆栈跟踪过滤器模式

9.2　HelloWorld 简单测试实例的开发

JUnit 框架可以将测试代码和代码完全分开，按照 TDD 的规则，应该在代码建立之前先把测试写好。这里假设未来的类名是 HelloWorld，并且有一个方法 say()，这个方法返回 String 的值（例如“HelloWorld！”）。

跟我做

（1）在 Eclipse 中创建 Java 工程“JUnitTest”，创建“src”源文件夹，并且在源文件夹中创建“test”包，如图 9-4 所示。

（2）在“test”包中创建 HelloWorld.java 类，并输入如下代码：

```
public class HelloWorld {

    public String say(){
        return "HelloWorld!";
    }
}
```

该类是经典的 HelloWorld 类，say 方法仅返回字符串“HelloWorld”。

（3）右击“test”包，在快捷菜单中选择【新建】|【JUnit 测试用例】命令，弹出【新建测试用例】对话框，提示 JUnit 库“junit.jar”不在构建路径上，如图 9-5 所示。

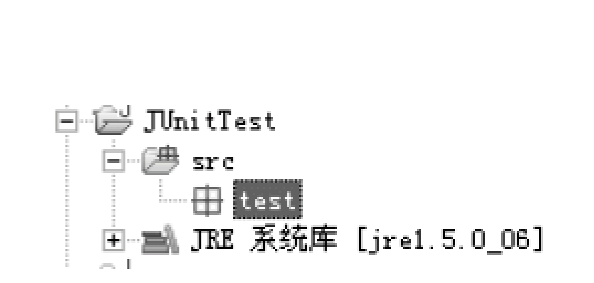


图 9-4　JUnitTest 工程结构

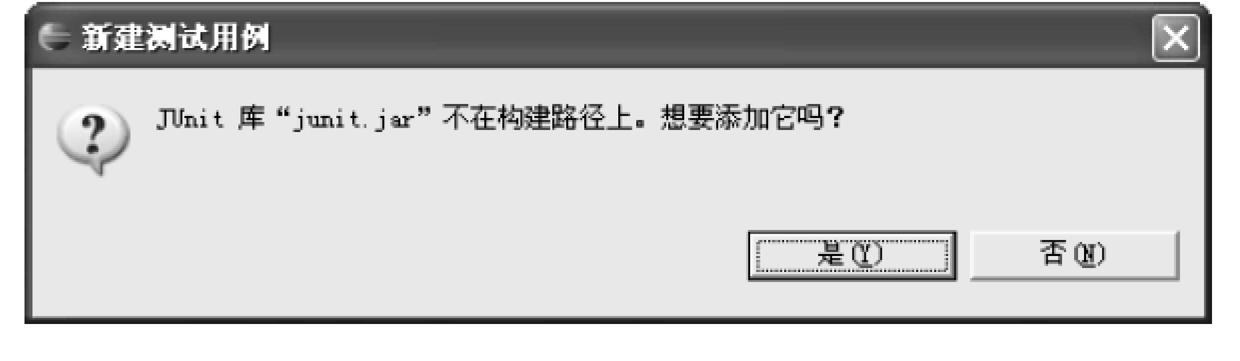


图 9-5　【新建测试用例】对话框

（4）单击【是】按钮，将 junit.jar 加到构建路径上，并打开【新建 JUnit 测试用例】对话框，如图 9-6 所示。



图 9-6　新建 JUnit 测试用例

（5）在【正在测试的类】文本框中选择“HelloWorld”类，在【名称】文本框中输入“TestSay”，单击【完成】按钮，生成如下代码：

```
package test;

import junit.framework.TestCase;

public class TestSay extends TestCase {

}
```

（6）编辑 TestSay.java 文件，输入如下代码：

```
    public void testSay(){
    //创建 HelloWorld 对象
        HelloWorld hi=new HelloWorld();
    //判断 say()方法的返回者是否为“Hello World！”
        assertEquals("Hello World!",hi.say());
    }
```

该方法首先创建 HelloWorld 对象，然后判断其 say()方法的返回值是否为字符串“Hello World！”。

（7）右击“TestSay.java”文件，在快捷菜单中选择【运行方式】|【JUnit 测试】命令，打开 JUnit 视图，如图 9-7 所示。

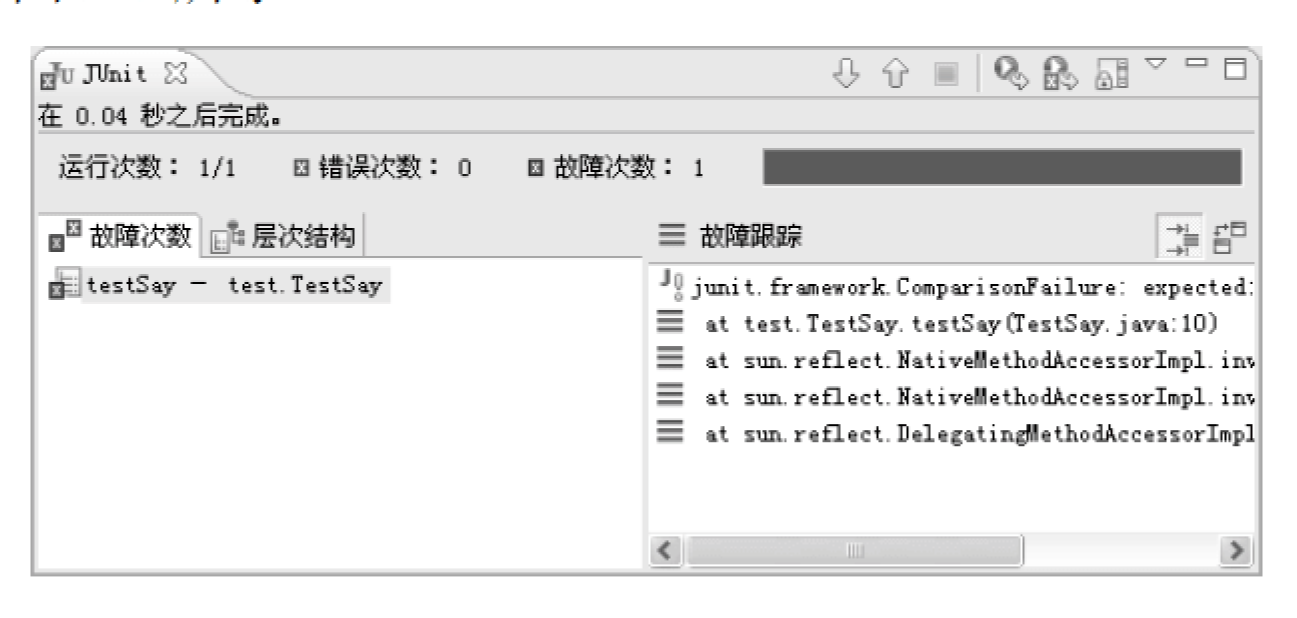


图 9-7　JUnit 视图

JUnit 视图包含运行次数、错误次数、故障次数、故障跟踪等信息。JUnit 视图有两个选项卡：一个显示故障列表，另一个将整个测试套件显示为一颗树。可以通过双击故障跟踪中的相应行来从故障浏览至相应的源。

（8）单击按钮，将实际的测试结果与期望的测试结果进行比较，弹出【比较结果】窗口，如图 9-8 所示。



图 9-8　【比较结果】窗口

通过比较期望结果与实际结果发现存在差异，所以测试失败。

（9）修改 TestSay.java 文件如下：

```
package test;


import junit.framework.TestCase;

public class TestSay extends TestCase {

	public void testSay(){
		HelloWorld hi=new HelloWorld();
		assertEquals("HelloWorld!",hi.say());
	}

}
```

即将“Hello World！”中的空格去掉，修改为“HellWorld!”。

（10）右击“TestSay.java”文件，在快捷菜单中选择【运行方式】|【JUnit 测试】命令，打开 JUnit 视图，如图 9-9 所示，测试成功。



图 9-9　JUnit 视图

9.3　创建测试用例

通过 JUnit 测试生成器，可以创建在单元测试中用作框架的可编译测试类。可以为单个类和整个包创建单元测试，也可以创建空的测试框架用于以后创建的源。可以通过在测试类的名称后面附加 Test 来区分生成的测试（如 MyClassTest.java）。

第 8 章的图书管理系统其核心是通过 Hibernate 对数据库进行查询、插入、更新等操作。本节介绍对这几个核心数据库操作创建测试用例的详细步骤。

跟我做

（1）在“Library”工程的“library.test”包中创建 DBOperation.java 类。编辑该类，输入如下代码：

```
package library.test;

import java.util.List;
```

```
import library.hibernate.UserTable;
import library.main.HibernateUtil;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;

/**
 * 通过 Hibernate 访问数据库的插入、更新、删除和查询操作
 *
 * @author lianhw
 *
 */
public class DBOperation {

	/**
	 * 数据库的查询操作
	 *
	 * @param hql
	 *            查询语句
	 * @return 查询返回的结果集
	 */
	public List select(String hql) {
		// 创建 SessionFactory 对象
		SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
		// 从会话工厂类中打开一个会话
		Session session = sessionFactory.openSession();
		// 创建一个事务
		Transaction tx = session.beginTransaction();
		// 执行查询操作
		Query userList = session.createQuery(hql);
		// 将查询结果放到一个列表中
		List list = userList.list();
		// 提交事务
		tx.commit();
		// 关闭会话
		session.close();
		// 返回结果
		return list;
	}

	/**
	 * 数据库的插入和更新操作
	 *
	 * @param userTable
	 */
	public void saveOrUpdate(UserTable userTable) {
```

```
        // 创建 SessionFactory 对象
        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        // 从会话工厂类中打开一个会话
        Session session = sessionFactory.openSession();
        // 创建一个事务
        Transaction tx = session.beginTransaction();
        // 插入和更新操作
        session.saveOrUpdate(userTable);
        // 提交事务
        tx.commit();
        // 关闭会话
        session.close();
    }

    /**
     * 数据库的删除操作
     *
     * @param userTable
     */
    public void delete(UserTable userTable) {
        // 创建 SessionFactory 对象
        SessionFactory sessionFactory = HibernateUtil.getSessionFactory();
        // 从会话工厂类中打开一个会话
        Session session = sessionFactory.openSession();
        // 创建一个事务
        Transaction tx = session.beginTransaction();
        // 数据库的删除操作
        session.delete(userTable);
        // 提交事务
        tx.commit();
        // 关闭会话
        session.close();
    }
}
```

该类实现了数据的插入操作、更新操作、删除操作和查询操作。

（2）右击“test”包，在快捷菜单中选择【新建】|【JUnit 测试用例】命令，打开【新建 JUnit 测试用例】对话框。

（3）在【名称】文本框中输入“TestDBSaveOrUpdate”，在【正在测试的类】文本框中选择“library.test.DBOperation”，如图 9-10 所示。

（4）单击【完成】按钮，创建 TestDBSaveOrUpdate.java 类。编辑该类，输入如下信息：



图 9-10　新建 JUnit 测试用例

```
package library.test;

import java.util.List;

import junit.framework.TestCase;
import library.hibernate.UserTable;

/**
 * 测试数据库操作的用例
 *
 * @author lianhw
 *
 */
public class TestDBSaveOrUpdate extends TestCase {

	/**
	 * 测试数据库的插入、查询、删除和更新操作
	 */
	public void testDBOperation() {
		//创建 DBOperation 对象
		DBOperation db = new DBOperation();
		//创建 UserTable 对象
		UserTable userTable = new UserTable();
		//设置 UserName 属性
		userTable.setUserName("王燕");
		//设置 password 属性
		userTable.setPassword("123456");
		//设置 Power 属性
		userTable.setPower("图书管理员");
		//执行插入或更新操作
		db.saveOrUpdate(userTable);
		//查询语句
		String hql = "from UserTable where UserName='王燕' and Password='123456'";
		//执行查询语句
		List result = db.select(hql);
		//判断返回的结果不为 null
		assertNotNull(result);
		//判断返回的结果包含刚插入的对象
		assertEquals("王燕", ((UserTable) result.get(0)).getUserName());
		//删除特定的对象
		db.delete(userTable);
		//查询已经删除的对象
		result = db.select(hql);
		//判断返回结果的链表是否为空
		assertEquals(result.size(), 0);
	}
}
```

该测试用例首先将一个 UserTable 实例插入到数据库中，插入后查询，判断返回的查询

结果中是否包含刚才插入的实例，从而据此判断插入或更新操作是否成功。最后又将新查询的记录删除，这样再次作查询其返回结果应该不包含该类。

（5）右击“TestSay.java”文件，在快捷菜单中选择【运行方式】|【JUnit 测试】命令，打开 JUnit 视图，如图 9-9 所示，测试成功。

（6）修改一条判断语句，将

```
assertEquals("王燕", ((UserTable) result.get(0)).getUserName());
```

修改成

```
assertEquals("王", ((UserTable) result.get(0)).getUserName());
```

（7）再次运行该测试用例，JUnit 视图的错误报告如图 9-11 所示。



图 9-11　JUnit 视图的错误报告

该图表明测试用例没有通过，并且指明了错误发生的原因、位置等信息，根据这些信息进行修改，直到测试通过为止。

9.4　创建测试套件

在 JUnit 中，Test、TestCase 和 TestSuite 三者组成了 composiste pattern。通过组装自己的 TestSuite，可以完成对添加到这个 TestSuite 中所有 TestCase 的调用。而且这些定义的 TestSuite 还可以组装成更大的 TestSuite，这样同时也方便了对于不断增加的 TestCase 的管理和维护。它的另一个好处就是，可以从这个 TestCase 树的任意一个节点（TestSuite 或 TestCase）开始调用，来完成这个节点以下所有 TestCase 的调用，提高了单元测试的灵活性。

JUnit 的测试套件向导帮助创建测试套件。可以选择一组应当属于某个套件的类。本节介绍如何创建测试套件。

跟我做

（1）右击“Library”工程的“library.test”包，在快捷菜单中选择【新建】|【其他】命令，打开【新建】对话框，如图 9-12 所示。

（2）在【新建】对话框中选择【JUnit 测试套件】选项，单击【下一步】按钮，打开【新建 JUnit 测试套件】对话框。

（3）在【名称】文本框中保持默认的“AllTests”，当前只有一个测试类，但是以后可

以对套件进行添加，如图 9-13 所示。




图 9-12　【新建】对话框



图 9-13　新建 JUnit 测试套件

（4）单击【完成】按钮，生成 AllTests.java 文件，其内容如下：

```
package library.test;

import junit.framework.Test;
import junit.framework.TestSuite;

public class AllTests {

	public static Test suite() {
		TestSuite suite = new TestSuite("Test for library.test");
		//$JUnit-BEGIN$
		suite.addTestSuite(TestDBSaveOrUpdate.class);
		//$JUnit-END$
		return suite;
	}

}
```

注意： 向导将两个标记（//$JUnit-BEGIN$ 和 //$JUnit-END$）放到已创建的测试套件类中，这允许向导更新现有的测试套件类。不建议在两个标记之间编辑代码。

（5）右击“AllTests.java”文件，在快捷菜单中选择【运行方式】|【JUnit 测试】命令，将执行该测试套件中的所有测试用例。其结果也同样反映到 JUnit 视图中。

9.5　定制测试配置与测试故障

本节介绍如何为测试传递参数或定制测试的运行。包括指定要运行的测试、其自变量、运行时类路径和 Java 运行环境。单元测试的目的就是快捷地定位代码中的错误。在发生测

试故障时，可以遵循下列步骤来调试它，修改错误后继续调试，直到正确无误为止。

跟我做

（1）单击工具栏中的 按钮，在快捷菜单中选择【运行…】命令，打开如图 9-14 所示的【运行】对话框。



图 9-14　【运行】对话框

（2）在【运行】对话框中指定要运行的测试、其自变量、运行时类路径和 Java 运行时的环境。

（3）单击【运行】按钮后如果出现故障条目，在 JUnit 视图的【故障】选项卡中双击故障条目以便在编辑器中打开相应的文件。

（4）在测试方法的开头设置断点。

（5）右击测试用例，在快捷菜单中选择【调试方式】|【JUnit 测试】命令。

第 10 章　AOP 开发实例

面向方面编程（Aspect-Oriented Programming，AOP）是一个令人兴奋的新模式，它的影响力将会和面向对象编程（OOP）一样。面向方面编程和面向对象编程并非相互竞争的技术而是可以很好地互补。面向对象编程主要用于为同一对象层次的公用行为建模。其弱点是如何将公共行为应用于多个无关对象模型之间。而这恰恰是 AOP 适合的地方。AOP 允许定义交叉的关系，这些关系应用于不同的对象模型。AOP 允许层次化功能性而不是嵌入功能性，使得代码有更好的可度性和易于维护性。OOP 是自上而下的软件开发，而 AOP 是自左而右的软件开发，它们是完全直交的技术，可以相互很好地补充。

本章介绍如何在 Eclipse 中进行 JBoss AOP 实例的开发，内容包括 JBossAOP 插件的安装和多个拦截器的实例编写。

10.1　AOP 术语解析

AOP 是 OOP 的延续。AOP 实际是 GoF 设计模式的延续，设计模式孜孜不倦追求的是调用者和被调用者之间的解耦，AOP 可以说也是这种目标的一种实现。在 OOP 的工具中是继承、封装和多态，而 AOP 的组件是指示/拦截器、引导、元数据等。本节首先对 AOP 的指示/拦截器、引导、元数据和切分点等术语进行解析。

10.1.1　指示/拦截器

指示就是一段有特定事件触发的程序逻辑，是能够被插入在调用者（激活其他方法的主体）和被调用者（被激活的方法）之间的行为。可以将日志和观测之类的功能运用在现有的对象模型上而不必过问实现细节。在 JBoss AOP 中，可以用拦截器来实现指示。可以定义拦截器来拦截方法调用、构造器调用和域访问。

10.1.2　引导（introduction）

利用引导，可以将方法或域增加到一个现有的类中。甚至允许修改某个现有类目前实现的接口，引入一个实现了那些新接口的混合类。引导将多继承特性注入到普通的 Java 类。

10.1.3　元数据

元数据是另一种能够追加到现有类之中的信息，可以在静态状态下或者运行期追加。

对于元数据的应用，一个很好的类比，就是 EJB 规范。在 EJB 的 XML 部署描述符中，会针对每一个方法分别定义事务属性。应用服务器于是知道应该在什么时候和什么地方开始、挂起或者提交一个事务。

10.1.4　切分点

切分点是粘合剂，告诉 AOP 框架，哪些拦截器绑定到哪些类，哪些元数据将应用于哪些类，或者哪一个引导被引入哪些类。切分点决定了各种 AOP 特征将怎样被运用于应用程序的类中。

10.2　下载并安装 JBossAOP 插件

JBossAOP 插件可以通过一个图形化用户界面为 Eclipse 工程定义拦截器，然后在 Eclipse 中运行该工程。

跟我做

（1）启动 Eclipse，单击【Help】菜单，选择【Software Updates】|【Find and Install】命令，打开【Install/Update】对话框。

（2）选择【Search for new features to install】选项，单击【下一步】按钮，进入【Install】对话框。单击【New Remote Site】按钮，打开【New Update Site】对话框。

（3）在【Name】文本框中输入“JBossIDE”，在 URL 文本框中输入“http://download.jboss.org/jbosside/updates/stable”，如图 10-1 所示。

（4）单击【OK】按钮，将名字为“JBossIDE”的新站点加入到列表中，如图 10-2 所示。



图 10-1　【New Update Site】对话框

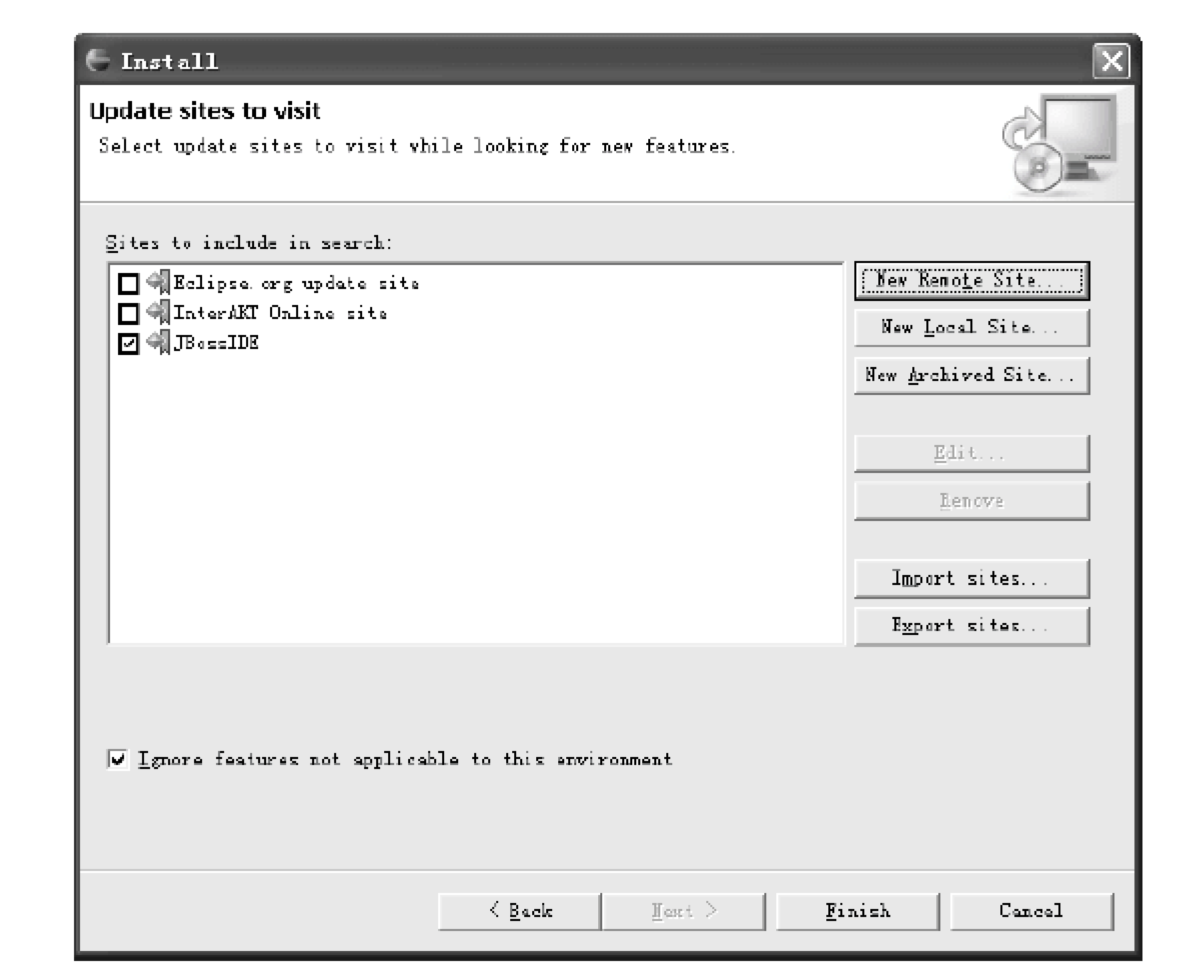


图 10-2　【Install】对话框

（5）单击【Finish】按钮，连接成功后打开【Updates】对话框，选择【JBoss AOP Tools1.1.2.GA】选项，如图 10-3 所示。

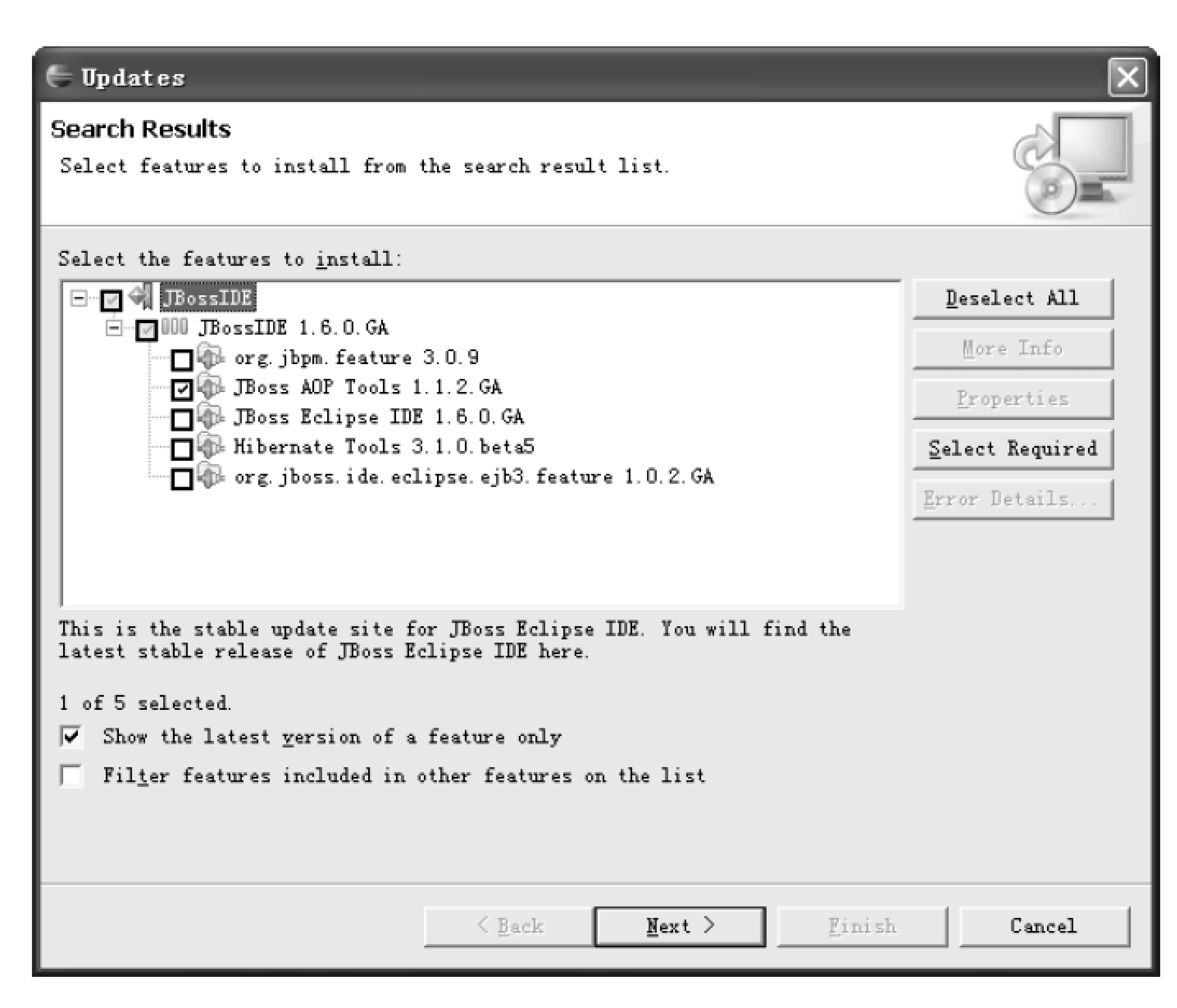


图 10-3　【Updates】对话框

（6）单击【Next】按钮，选择【I accept the terms in the license agreement】选项，单击【Next】按钮，进入如图 10-4 所示对话框。

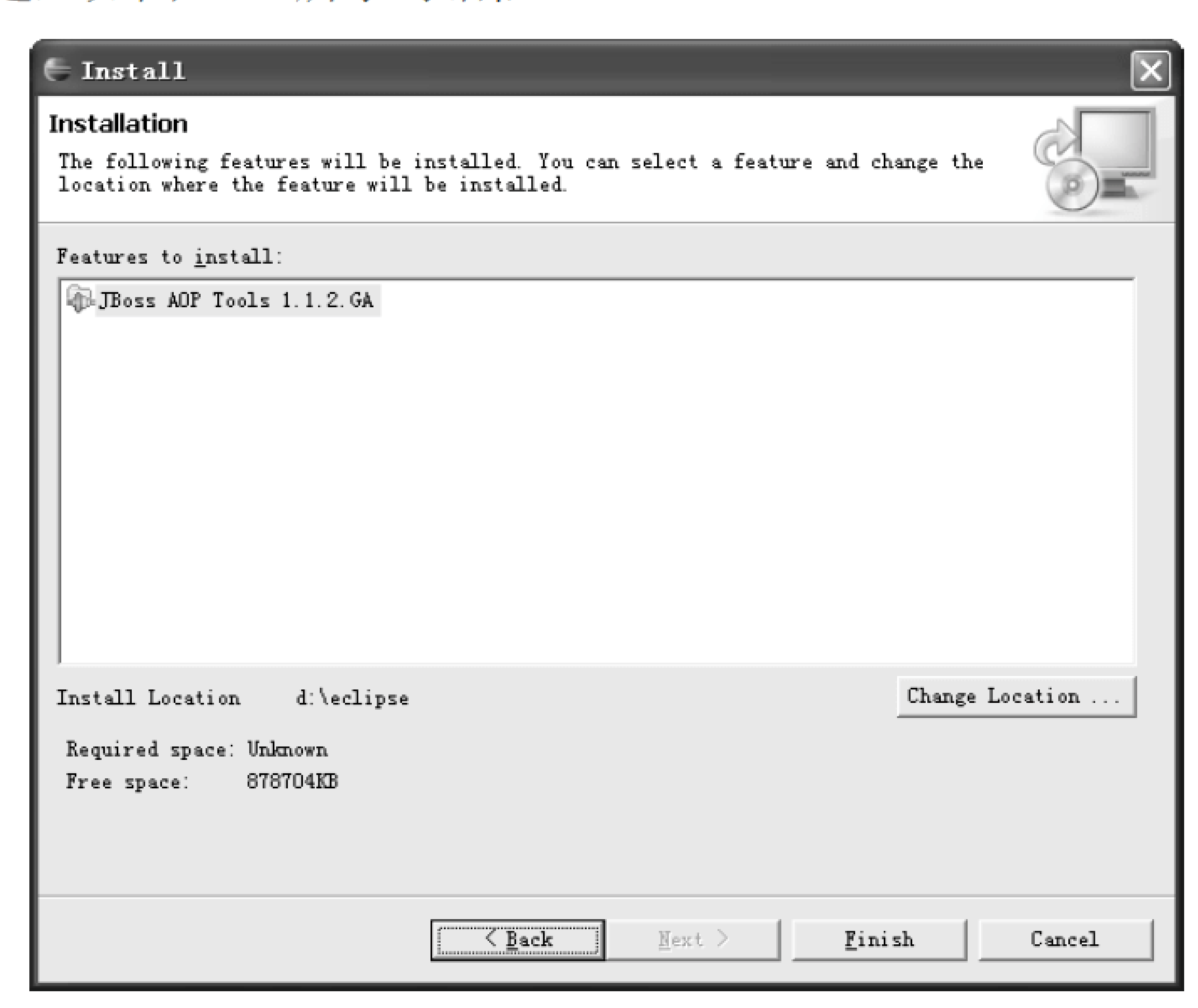


图 10-4　【Install】对话框

（7）单击【Finish】按钮，完成 JBossAOP 插件的安装。

10.3　第一个 AOP 实例

JBossAOP 框架和 JBoss 应用服务器紧密地结合，但也可以在自己的应用程序中独立运

行。要理解一个概念，最好的办法莫过于看看它的实际应用，所以下面用一个 JBossAOP 中的例子，来说明所有这些部分是如何协同工作的。本节介绍如何在 Eclipse 中创建第一个 AOP 应用实例，内容包括创建 POJO、创建拦截器和生成 jboss-aop.xml 等。

10.3.1　编写 POJO

POJO 有一些 private 的参数作为对象的属性，然后针对每个参数定义了 get 和 set 方法作为访问的接口。POJO 对象有时也被称为 Data 对象，大量应用于表现现实中的对象。

本小节创建一个经典的 HelloWorld 级别的 POJO，向控制台中输出一个字符串。

跟我做

（1）启动 Eclipse，单击【文件】菜单，选择【新建】|【项目】命令，打开【新建项目】对话框，选择【JBossAOP Project】选项，如图 10-5 所示。

（2）单击【下一步】按钮，在【项目名】文本框中输入“HelloAOP”，如图 10-6 所示。




图 10-5　新建 JBossAOP 项目



图 10-6　HelloAOP 工程

（3）单击【完成】按钮，生成 HelloAOP 工程，如图 10-7 所示。



图 10-7　HelloAOP 工程

（4）右击“HelloAOP”工程的“src”文件夹，在快捷菜单中选择【新建】|【类】命

令，创建 HelloAOP.java 类。

（5）编辑 HelloAOP.java 类，输入如下代码。

```
public class HelloAOP {

	/**
	 * 向控制台输出“AOP!”字符串
	 */
	public void callMe() {
		System.out.println("AOP!");
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
		new HelloAOP().callMe();

	}
}
```

10.3.2　编写拦截器

在 JBossAOP 中是用拦截器来实现 advice 的。可以自定义拦截器、拦截方法调用、构造函数调用以及对字段的访问，但 JBoss 要求这些自定义的拦截器必须实现 org.jboss.aop.Interceptor 接口：

```
public interface Interceptor
{
	public String getName();
	public InvocationResponse invoke(Invocation invocation) throws Throwable;
}
```

在 JBossAOP 中，被拦截的字段、构造器和方法均被转化为通用的 invoke 方法调用。方法的参数接收一个 Invocation 对象，而方法的返回值、字段的存取以及构造函数则被填入一个 InvocationResponse 对象。Invocation 对象同时还驱动拦截链。所有的拦截器需要继承 org.jboss.aop.advice.Interceptor 类。本小节创建一个拦截器，向控制台输出“Hello”字符。

跟我做

（1）右击“HelloAOP”工程的“src”文件夹，在快捷菜单中选择【新建】|【类】命令，打开【新建 Java 类】对话框，在【名称】文本框中输入“HelloAOPInterceptor”，让该类实现 Interceptor 接口，如图 10-8 所示。



图 10-8　新建拦截器

（2）单击【完成】按钮，创建 HelloAOPInterceptor.java 类。编辑该类，输入如下代码：

```
import org.jboss.aop.advice.Interceptor;
import org.jboss.aop.joinpoint.Invocation;

public class HelloAOPInterceptor implements Interceptor {

    /**
     * 返回拦截器的名字
     * @see org.jboss.aop.advice.Interceptor#getName()
     */
    public String getName() {
        return "HelloAOPInterceptor";
    }


    /**
     * 拦截器的 invoke 方法
     * @see org.jboss.aop.advice.Interceptor#invoke(org.jboss.aop.joinpoint.Invocation)
     */
    public Object invoke(Invocation invocation) throws Throwable {
        // 输出 Hello
        System.out.print("Hello, ");
        // 调用下一个方法
        return invocation.invokeNext();
    }

}
```

该拦截器输出“Hello”字符串。

10.3.3　将拦截器引用到 callMe()方法中

在 JBossAOP 插件中的方面管理器是 jboss-aop.xml 文件的图形化显示，可以减少由于缺乏静态检查而带来的问题（例如手工编辑 XML 的需要）。它还对程序的横切结构提供了方便的完整系统显示。可以快捷地生成 jboss-aop.xml 文件，本小节介绍如何将拦截器应用到 callMe)()方法中。

跟我做

（1）打开 HelloAOP.java 编辑器，在大纲视图中可以看到 callMe()方法，如图 10-9 所示。

（2）右击 callMe()方法，在弹出的快捷菜单中选择【JBoss AOP】|【Apply interceptor】命令，出现【Select an Interceptor】界面，如图 10-10 所示。



图 10-9　大纲视图

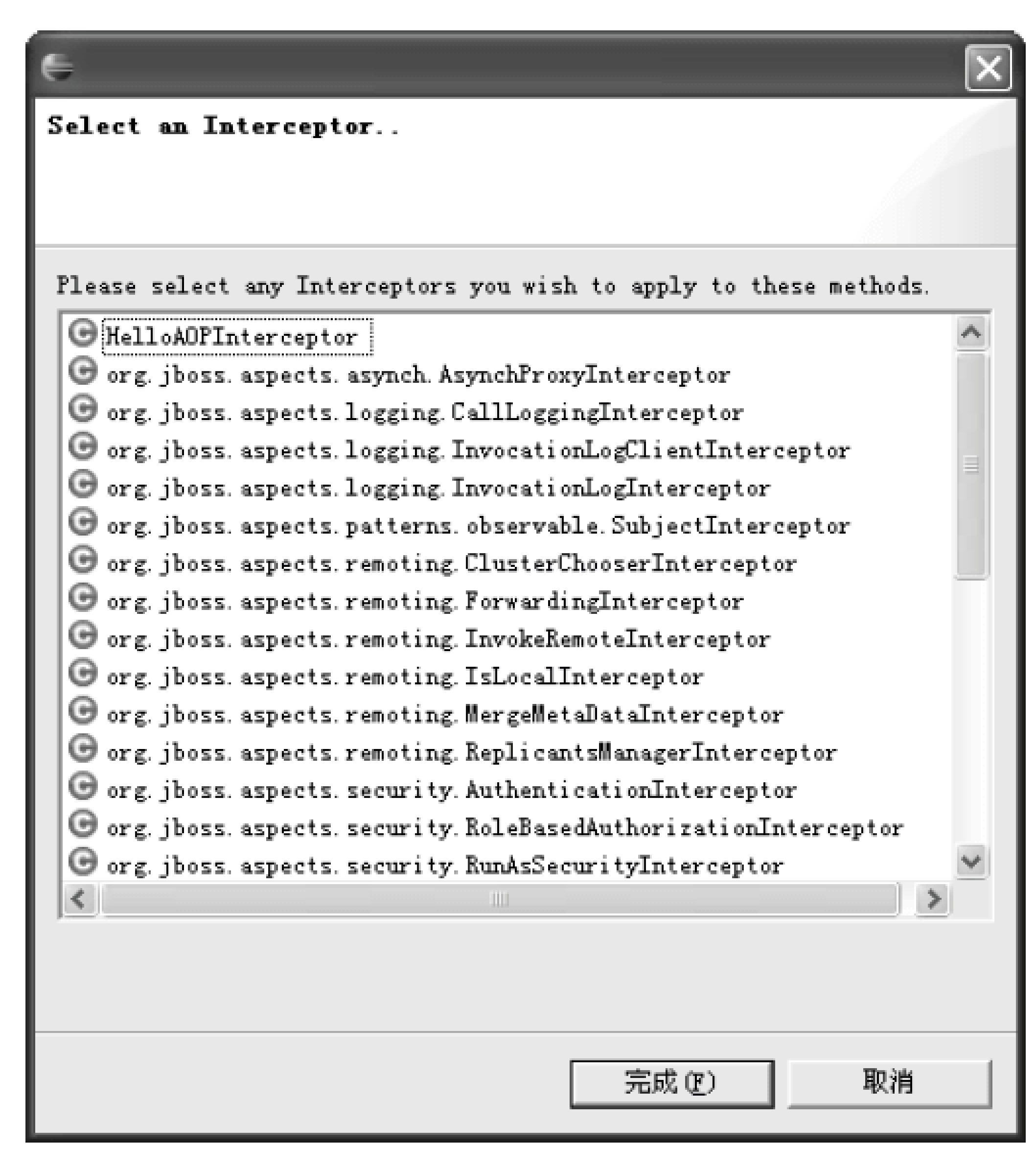


图 10-10　选择拦截器

（3）选择【HelloAOPInterceptor】，单击【完成】按钮，更新 jboss-aop.xml 文件的内容如下所示：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut="execution(public void HelloAOP-&gt;callMe())">
        <interceptor class="HelloAOPInterceptor"/>
    </bind>
</aop>
```

该配置文件说明了将 HelloAOPInterceptor 拦截器应用到 callMe()方法中。

10.3.4　运行实例

本小节运行 10.3 节实现的实例，看一看拦截器是否起作用了。

跟我做

（1）单击工具栏中的【运行...】按钮，打开【运行】对话框。

（2）双击【JBoss AOP Application】选项，生成一个默认名字为“新建配置”的运行配置。将其名称修改为“HelloAOP”，【项目】选择“HelloAOP”工程，【Main 类】选择“HelloAOP”类，如图 10-11 所示。

（3）单击【运行】按钮，在控制台上输出如图 10-12 所示内容。



图 10-11　【运行】对话框



图 10-12　控制台的输出

10.4　属性拦截实例

通过判断拦截器的 invoke（Invocation invocation）方法的参数 invocation 是 FieldReadInvocation 还是 FieldWriteInvocation 来判断当前进行的是属性的读操作还是写操作。同时通过 Invocation 对象的 getField()方法可以取得当前操作的具体属性。本节介绍如何为每个类的属性加入拦截器，从而跟踪对该属性的访问。

跟我做

（1）编辑 HelloAOP.java 文件，为其加入字符串类型的属性 str，并且创建该属性的 setter/getter 方法。代码如下所示：

```
public class HelloAOP {

    // 字符串属性 str
```

```
	private String str = "AOP!";

	/**
	 * 向控制台输出“AOP!”字符串
	 */
	public void callMe() {
		System.out.println("AOP!");
	}

	/**
	 * 取得 str 属性
	 * @return
	 */
	public String getStr() {
		return str;
	}

	/**
	 * 设置 str 属性
	 * @param str
	 */
	public void setStr(String str) {
		this.str = str;
	}

	/**
	 * @param args
	 */
	public static void main(String[] args) {
	HelloAOP hello=new HelloAOP();
	// hello.callMe();
		System.out.println(hello.getStr());
		hello.setStr("newAOP!");

	}
}
```

（2）右击“HelloAOP”工程的“src”源文件夹，创建“HelloAOPFieldInterceptor.java”文件，该类继承于 org.jboss.aop.advice.Interceptor 类。编辑该类，输入如下代码：

```
import org.jboss.aop.advice.Interceptor;
import org.jboss.aop.joinpoint.FieldReadInvocation;
import org.jboss.aop.joinpoint.FieldWriteInvocation;
import org.jboss.aop.joinpoint.Invocation;

public class HelloAOPFieldInterceptor implements Interceptor {

	/**
	 * 返回拦截器的名字
```

```
     *
     * @see org.jboss.aop.advice.Interceptor#getName()
     */
    public String getName() {
        return "HelloAOPFieldIntreceptor";
    }

    /**
     * 拦截器的 invoke 方法
     *
     * @see org.jboss.aop.advice.Interceptor#invoke(org.jboss.aop.joinpoint.Invocation)
     */
    public Object invoke(Invocation invocation) throws Throwable {
        // 判断是否为读取属性值操作
        if (invocation instanceof FieldReadInvocation) {
            // 输出提示信息
            System.out.println("调用属性 "
                    + ((FieldReadInvocation) invocation).getField()
                    + " 的 read 方法");
        }
        // 判断是否为属性设值
        else if (invocation instanceof FieldWriteInvocation) {
            // 输出提示信息
            System.out.println("调用属性 "
                    + ((FieldWriteInvocation) invocation).getField()
                    + " 的 write 方法");
        }
        return invocation.invokeNext();
    }

}
```

该拦截器判断当前进行的属性操作类型，从而给出相应的提示信息。当拦截器被调用后判断是否属于 FieldReadInvocation 或 FieldWriteInvocation，从而执行相应的操作。

（3）打开 HelloAOP.java 文件的大纲视图，右击“str”属性，在快捷菜单中选择【JBoss AOP】|【Apply interceptor】命令，出现【Select an Interceptor】界面，如图 10-13 所示。

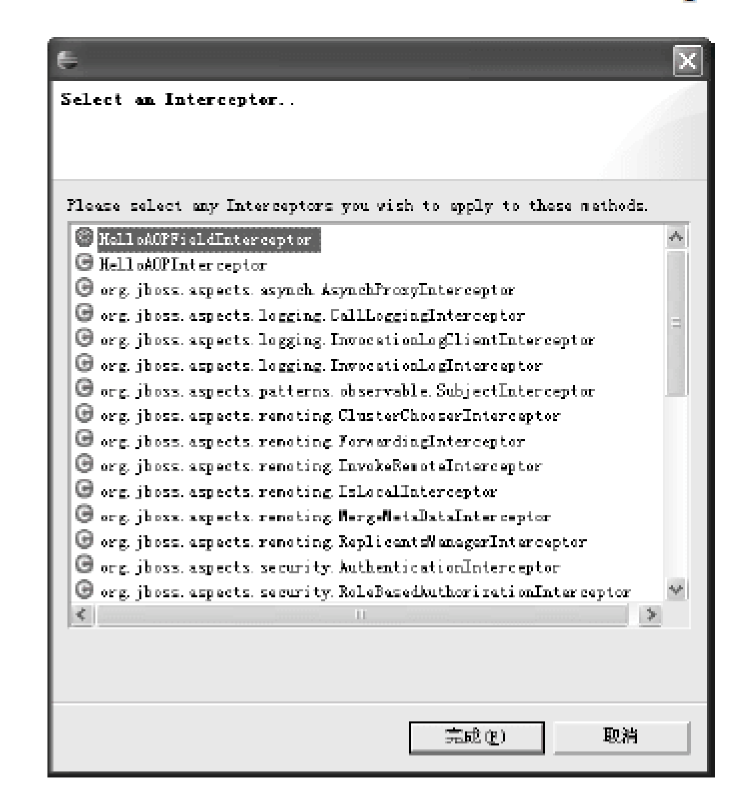


图 10-13　选择拦截器

（4）选择“HelloAOPFieldInterceptor”，单击【完成】按钮，更新 jboss-aop.xml 文件的内容如下所示：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut="execution(public void HelloAOP-&gt;callMe())">
        <interceptor class="HelloAOPInterceptor"/>
    </bind>
    <bind pointcut="field(private java.lang.String HelloAOP-&gt;str)">
        <interceptor class="HelloAOPFieldInterceptor"/>
    </bind>
</aop>
```

（5）在 HelloAOP.java 的 main 方法中调用 str 属性的 setter/getter 方法，如下所示：

```
public static void main(String[] args) {
            HelloAOP hello=new HelloAOP();
//          hello.callMe();
            System.out.println(hello.getStr());
            hello.setStr("newAOP!");
     }
```

（6）运行 HelloAOP.java，其控制台输出信息如图 10-14 所示。

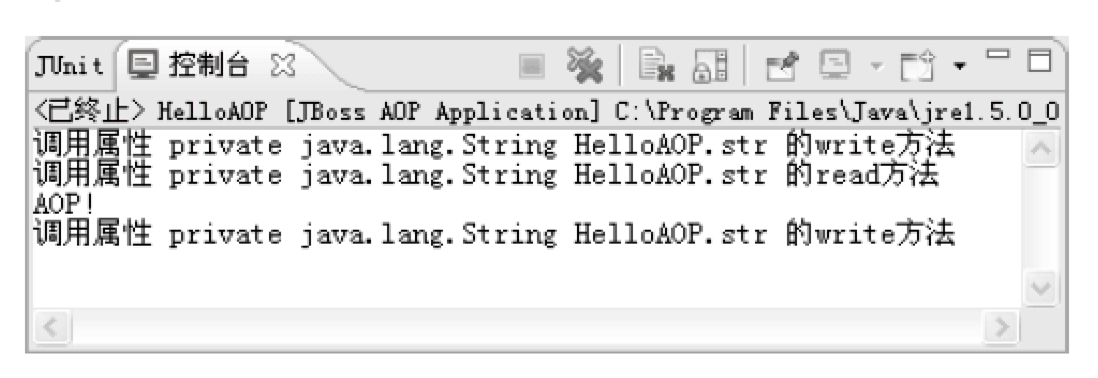


图 10-14　控制台输出信息

在 HelloAOP 类初始化时为 str 属性赋值，所以调用了该属性的 write 方法。hello.getStr() 调用了属性的 read 方法，将 getStr()方法执行的结果输出到控制台；hello.setStr（“newAOP!”）调用了属性的 write 方法，为属性设置了新值。

10.5　方法拦截实例

首先实现一个扩展 Inceptor 类的方法拦截器，然后在 jboss-aop.xml 文件中通过如下代码片断

```
<bind pointcut="execution(public static void POJO-&gt;test2())">
        <interceptor class="MethodInterceptor"/>
    </bind>
```

将连接器应用到 POJO 的所有方法中。本节介绍方法拦截实例，从而跟踪对类方法的访问。

跟我做

（1）在“HelloAOP”工程中创建 POJO.java 类，编辑该类输入如下代码：

```
public class POJO
{
    public void noop()
    {
        System.out.println("noop()");
    }

    public void test1(String param)
    {
        System.out.println("test1(String param): " + param);
    }

    public void test2(int param)
    {
        System.out.println("test2(int param): " + param);
        callBar();
    }

    public static void test2()
    {
        System.out.println("static method");
    }

    public void callBar()
    {
        new Bar().hello();
    }

    public class Bar
    {
        public void hello() { System.out.println("hello"); }
    }
}
```

该类定义了若干个方法，为了简单起见每个方法只是输出一些提示信息。

（2）创建 MethodInterceptor.java 文件，编辑该文件，输入如下信息：

```
public class MethodInterceptor implements Interceptor
{
    //取得拦截器的名字
    public String getName() { return "MethodInterceptor"; }

    //拦截器的 invoke 方法
    public Object invoke(Invocation invocation) throws Throwable
    {
        try
        {
            //将 invocation 进行类型转换
            MethodInvocation mi = (MethodInvocation)invocation;
            System.out.println("<<< Entering MethodInterceptor for: " + mi.getMethod().toString());
```

```
            return invocation.invokeNext();
        }
        finally
        {
            System.out.println(">>> Leaving MethodInterceptor");
        }
    }
}
```

该拦截器在每个方法调用之前打印进入该方法提示信息，等到方法执行完成后再输出离开该方法的提示信息。

（3）将 MethodInterceptor 连接器应用到 POJO 的所有方法中，jboss-aop.xml 文件的内容如下所示：

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<aop>
    <bind pointcut="execution(public void POJO$Bar-&gt;hello())">
        <interceptor class="MethodInterceptor"/>
    </bind>
    <bind pointcut="execution(public static void POJO-&gt;test2())">
        <interceptor class="MethodInterceptor"/>
    </bind>
    <bind pointcut="execution(public void POJO-&gt;callBar())">
        <interceptor class="MethodInterceptor"/>
    </bind>
    <bind pointcut="execution(public void POJO-&gt;noop())">
        <interceptor class="MethodInterceptor"/>
    </bind>
    <bind pointcut="execution(public void POJO-&gt;test1(java.lang.String))">
        <interceptor class="MethodInterceptor"/>
    </bind>
    <bind pointcut="execution(public void POJO-&gt;test2(int))">
        <interceptor class="MethodInterceptor"/>
    </bind>
</aop>
```

（4）创建 Driver.java 文件，在该文件中调用 POJO 的所有方法：

```
public class Driver
{
    public static void main(String[] args)
    {
        //创建 POJO 实例
        POJO pojo = new POJO();
        System.out.println("--- pojo.noop(); ---");
        //调用 noop()方法
        pojo.noop();
        System.out.println("--- pojo.test1(String param); ---");
        //调用 test1()方法
        pojo.test1("hello world");
```

```
        System.out.println("--- pojo.test1(int param); ---");
        //调用 test2()方法
        pojo.test2(55);
        System.out.println("--- POJO.test2(); ---");
        //调用 POJO 的静态方法 test2
        POJO.test2();
    }
}
```

（5）运行该 AOP 实例，运行结果如图 10-15 所示。

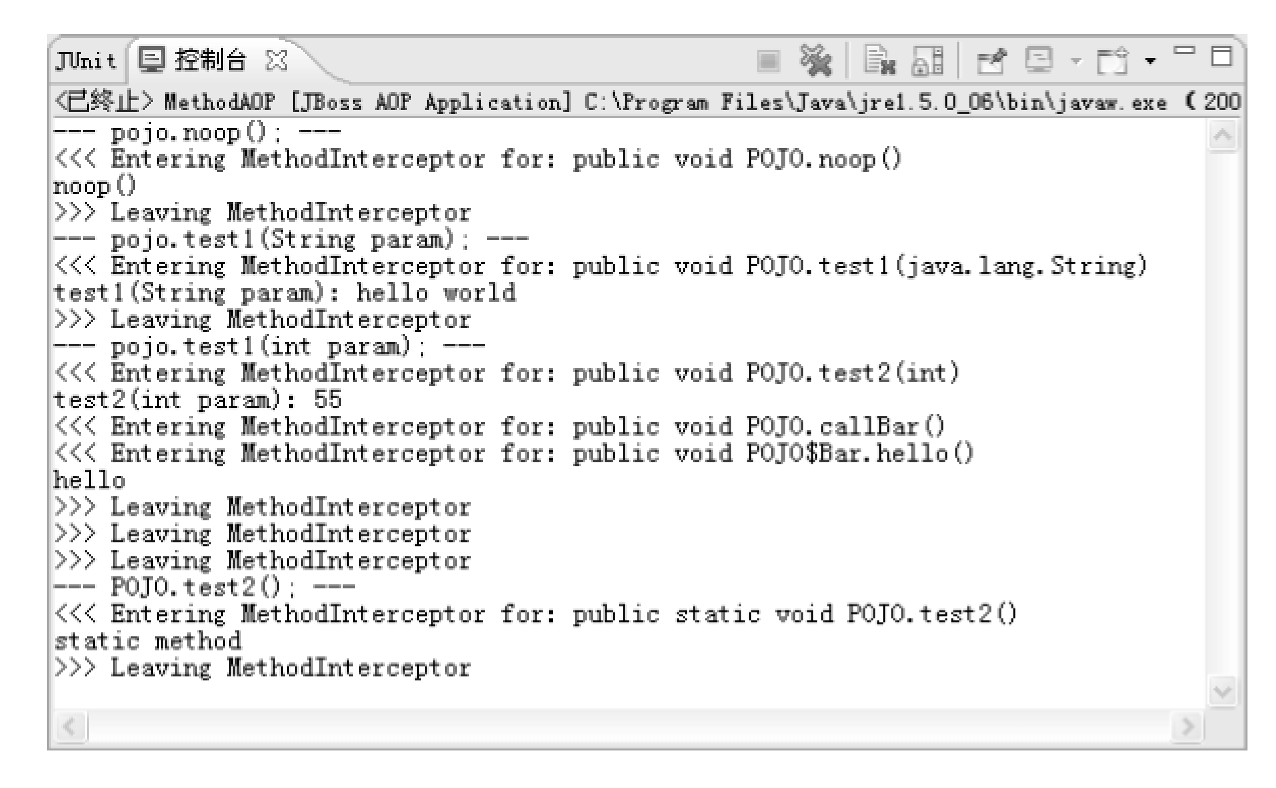


图 10-15　MethodInterceptor 的输出结果

第 11 章　在 Eclipse 中进行版本控制——CVS 使用实例

CVS 是 Concurrent Version System（并发版本系统）的简称。它是一个开放源代码的项目，是当前最流行的版本控制系统。CVS 采用客户机/服务器体系，代码、文档的各种版本都存储在服务器端，开发者首先从服务器上获得一份复制到本机，然后在此基础上进行开发。开发者可随时将新代码提交到服务器上，当然也可以通过更新操作获得最新的代码，保持与其他开发者的一致。

Eclipse 本身内置了 CVS 客户端，只要再建立一个 CVS 服务器就可以使用这一功能强大的版本控制系统了。本章介绍如何在 Eclipse 中进行版本控制，内容包括 CVS 服务器的安装和配置、使用 CVS 存储库共享本地项目和从现有的 CVS 存储库创建新项目。

11.1　下载并安装 CVS 服务器

CVS 起源于 Unix/Linux 平台，本节主要介绍 CVS 服务器的 Windows 版本 CVSNT 的安装和配置。

跟我做

（1）下载 cvsnt-2.5.01.1976.msi，运行该安装程序，不必更改它的任何默认设置，连续单击 next 按钮，即可完成安装。cvsnt 默认安装在 C:\Program Files\cvsnt 目录下。

（2）通过 Windows 选择【开始】|【所有程序】|【CVSNT】命令，打开【CVSNT】对话框，如图 11-1 所示。启动 CVS Service 和 CVS Lock service 两个服务器。

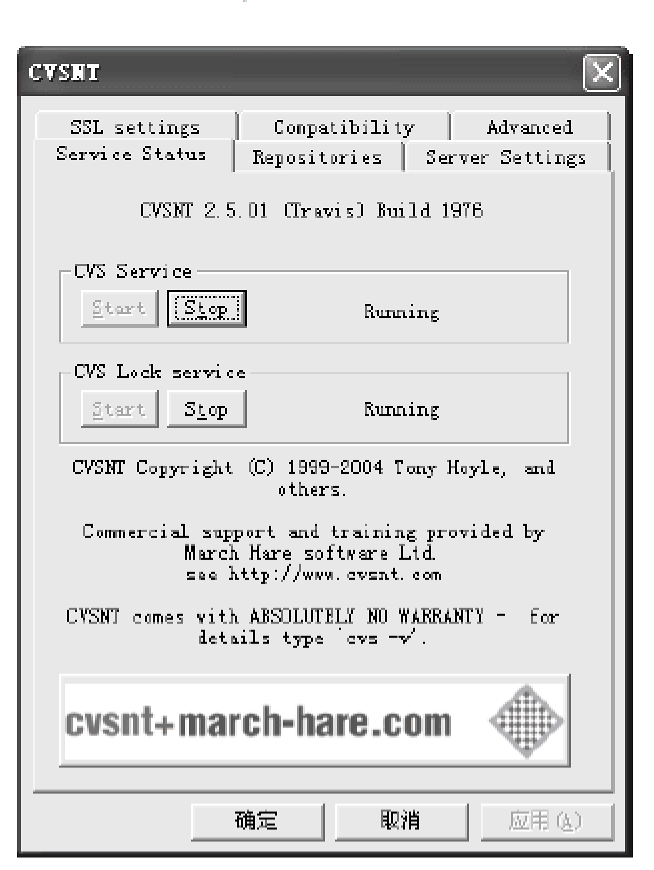


图 11-1　CVSNT 控制台

（3）选择【Repositories】选项卡，单击【Add】按钮，如图 11-2 所示。在【Location】文本框中选择“D:/cvsroot”，在【Name】文本框中输入“/cvsroot”，单击【OK】按钮，在服务器上创建了名字为“/cvsnt”的存储库，如图 11-3 所示。



图 11-2　【Repositories】选项卡

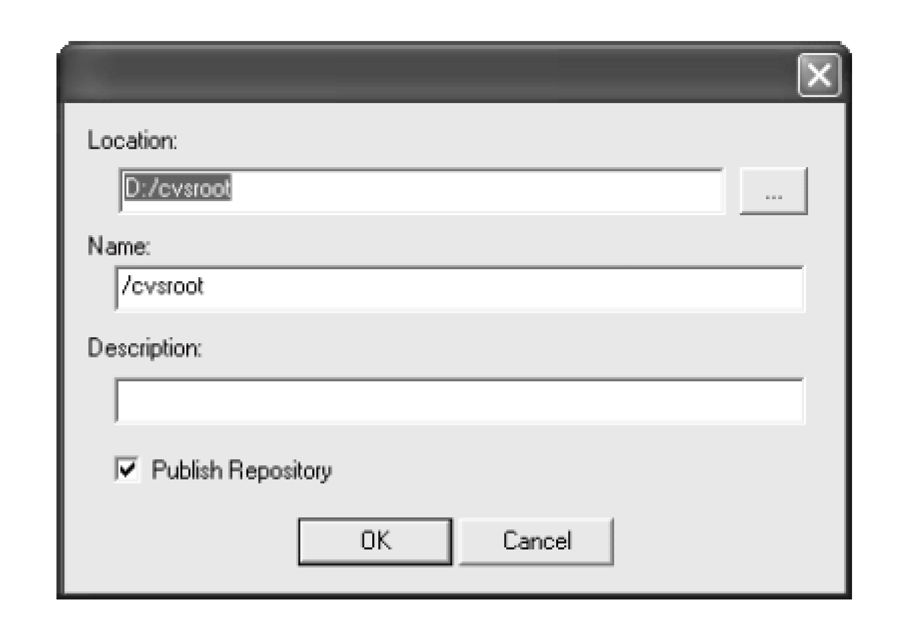


图 11-3　新增存储库

（4）在 Windows 的控制面板中选择【用户账号】创建一个新的用户“cvs”，密码为“cvs”，如图 11-4 所示。



图 11-4　创建新用户

至此，CVS 服务器的设置已经全部完成，客户端可以使用“cvs”为账号来登录该 CVS 服务器了。

11.2　在 Eclipse 中设置存储库

通过前面的安装与配置，已经正确安装了 CVS 服务器，并且取得了合法访问 CVS 服务器的用户名和密码，本节将学习如何在 Eclipse 中设置 CVS 存储库，从而利用 Eclipse 内置的 CVS 模块实现统一版本控制。

跟我做

（1）单击 Eclipse 的【窗口】菜单，选择【打开透视图】|【其他...】命令，打开【选择透视图】对话框，如图 11-5 所示。在该对话框中选择【CVS 存储库研究】选项，单击【确定】按钮，打开 CVS 存储库研究透视图。

（2）右击 CVS 存储库视图中的空白区域，在快捷菜单中选择【新建】|【存储库位置】命令，如图 11-6 所示。打开如图 11-7 所示的【添加 CVS 存储库】对话框。





图 11-5　选择 CVS 存储库透视图



图 11-6　新建存储库位置

（3）在【添加 CVS 存储库】对话框中进行如下配置：

- ❑ 在【主机】字段中输入 CVS 服务器主机的地址、域名或 IP 地址均可，例如输入“127.0.0.1”。
- ❑ 在【存储库路径】字段中输入主机地址处的存储库的路径，如“/cvsroot”。
- ❑ 在【用户】字段中输入用来进行连接的用户的名字“cvs”。
- ❑ 在【密码】字段中输入用来进行连接的用户的密码“cvs”。
- ❑ 【连接类型】字段选择“pserver”。
- ❑ 【使用缺省端口】保留为启用状态。
- ❑ 默认情况下会选择【在完成时验证连接】复选框。

（4）单击【完成】按钮。因为选择了完成时验证位置，所以向导现在将通过连接存储库来验证该信息。执行此操作时，可能会提示用户输入密码。完成后 CVS 存储库如图 11-8 所示。



图 11-7　添加 CVS 存储库

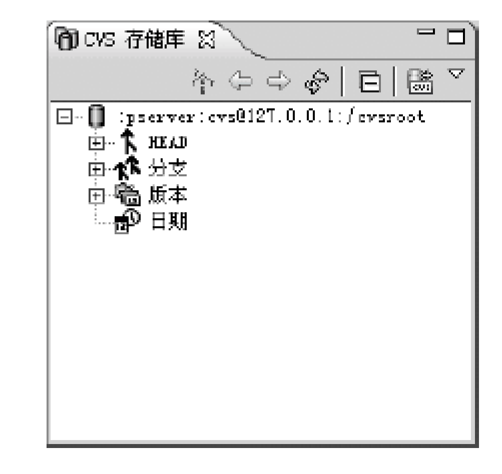


图 11-8　CVS 存储库视图

11.3　使用 CVS 存储库共享本地项目

每个新的空 Eclipse 项目都可以通过 CVS（或受支持的任何其他源代码管理系统）进行共享。开发人员也可以通过将其现有的代码迁移到资源库来共享它。要进行共享，单击项目主文件夹，在显示的上下文菜单中使用 Team/Share Project 选项即可。

通过前几节的配置，现在就可以将自己建立的工程发布到 CVS 服务器上，与开发团队的其他成员共享此工程了。

跟我做

（1）右击包资源管理器视图中的任意工程，比如“JavaApplication”工程，在快捷菜单中选择【小组】|【共享项目…】命令，打开【共享项目】对话框。

（2）在如图 11-9 所示的【共享项目】中从先前配置好的几个 CVS 存储库中选择正确的存储库位置，单击【下一步】按钮，打开如图 11-10 所示的对话框。

（3）在【输入模块名】界面中输入要在服务器上创建的模块名称。保持默认的【使用项目名称作为模块名称】。单击【下一步】按钮，打开【共享项目资源】对话框。

（4）在【共享项目资源】界面中可以查看将与开发团队共享的文件。带加号的箭头表示这些文件有新的传出添加内容。它们在服务器上尚未存在。单击【完成】按钮，如图 11-11 所示。




图 11-9　选择 CVS 存储库



图 11-10　输入模块名界面

（5）在【落实】界面中将询问是否要提交文件到 CVS 服务器上。可以为本次文件提交输入必要的注释信息，类似于一个文件提交的简单日志，完成后，单击【完成】按钮，文件将被提交到 CVS 服务器上，如图 11-12 所示。

至此完成了将本地 Java 工程共享到 CVS 服务器，这样开发团队中的其他人就可以从

CVS 服务器上检出该工程了。



图 11-11　共享项目资源

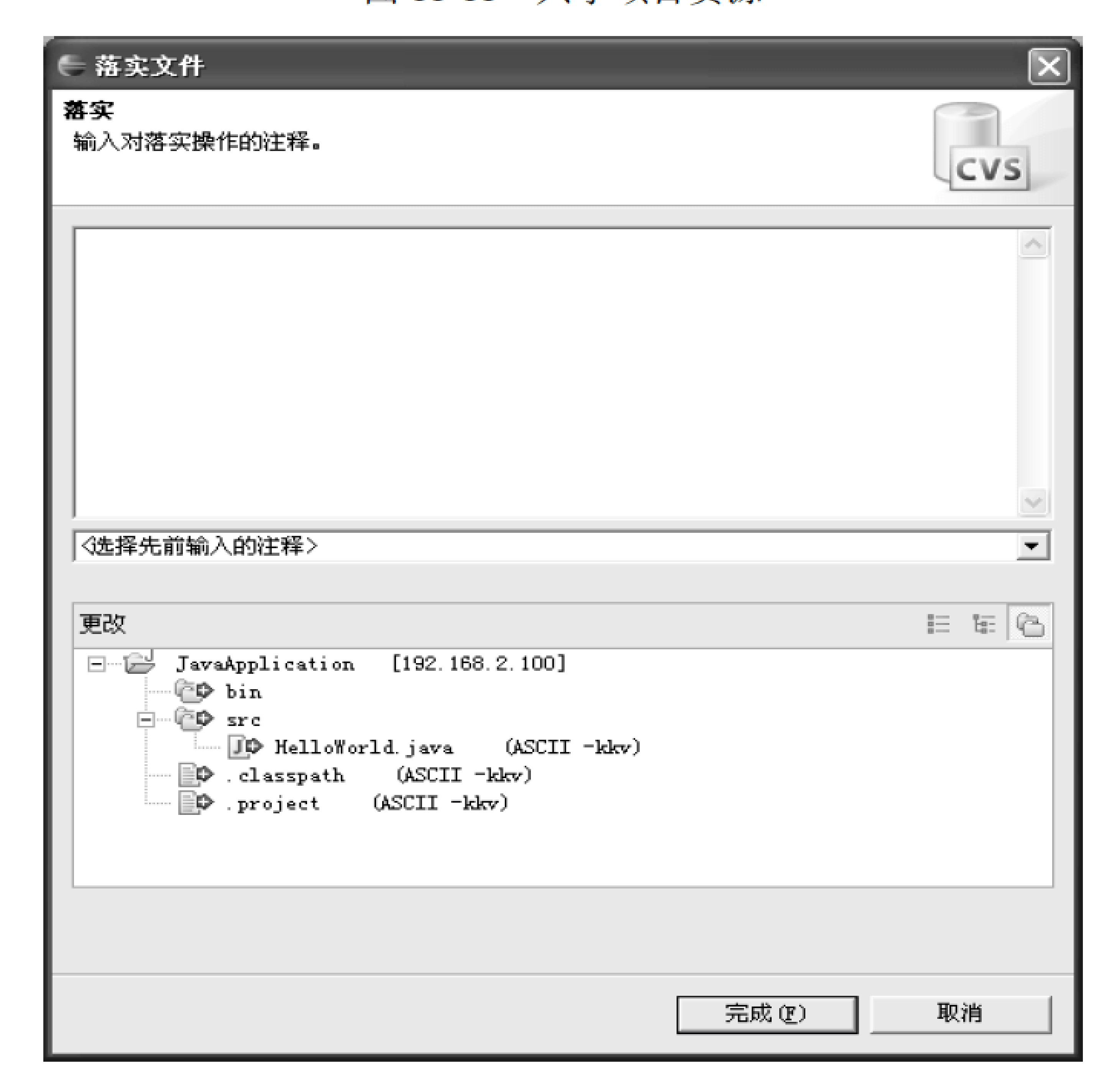


图 11-12　提交操作注释

11.4　从 CVS 服务器上检出已经存在的 Java 工程

通过从选定的 CVS 资源库分支导入代码来创建新的工作台项目。只要选择适当分支(或

HEAD），然后选择从 CVS Repository Exploring 透视图中的上下文菜单中选择【Checkout As Project】选项即可。本节介绍如何从 CVS 服务器上检出已经存在的 Java 工程。

跟我做

（1）打开 Eclipse 的 CVS 存储库研究透视图，在 CVS 存储库视图中单击“HEAD”前面的加号，可以看到 CVS 服务器上所有已经存在的 Java 工程。

（2）右击“JavaApplication”工程，在快捷菜单中选择【检出】命令。

（3）重新打开 Eclipse 的 Java 透视图，在包资源管理器中可以看到“JavaApplication”工程，如图 11-13 所示。

图 11-13　新检出的 JavaApplication 工程

11.5　使本地更改与 CVS 存储库同步

如果一个项目开发人员准备提交他 / 她的工作，那么首先要执行更新操作。这会针对引入的更改核对资源库，并将这些更改添加到该开发人员的本地工作台。这样确保了开发人员知道这些更改可能会影响他 / 她将要提交的工作的完整性。使用项目上下文菜单中的 Compare With...选项将本地版本与资源库中存储的代码进行比较。本节学习如何使本地更改与 CVS 存储库同步。

跟我做

（1）对 CvsUser1.java 文件作如下修改：

原始内容：

```
public class CvsUser1 {

      /**
       * @param args
       */
      public static void main(String[] args) {

      }
}
```

新内容：

```
       public class CvsUser1 {

      /**
```

```
        * @param args
        */
      public static void main(String[] args) {
            System.out.println(“第一个用户的修改”);

      }
}
```

修改后在 CvsUser1.java 类的前面就会出现一个“>”号，表示本地有了新的内容需要提交到 CVS 服务器上，如图 11-14 所示。

（2）右击“JavaApplication”工程，在快捷菜单中选择【小组】|【与存储库同步】命令，在同步视图中出现本地与服务器所有需要同步的项目，如图 11-15 所示。



图 11-14　修改 CvsUser1.java 后的情况



图 11-15　同步视图

（3）右击同步视图中的“CvsUser1.java”，在快捷菜单中选择【在比较编辑器中打开】命令，可以查看本地跟服务器端的内容差异，如图 11-16 所示。



图 11-16　比较编辑器

（4）右击同步视图中的“CvsUser1.java”，在快捷菜单中选择【小组】|【落实】命令，打开【落实文件】对话框，如图 11-17 所示。

（5）单击【完成】按钮，将文件提交到 CVS 服务器上。提交后，本地工程与服务器上的该工程保持了版本的一致。

（6）如果两个用户同时修改了 CvsUser1.java 文件，就会出现冲突，如图 11-18 所示。CvsUser1.java 文件前面的符号表示出现了冲突。



图 11-17　【落实文件】对话框

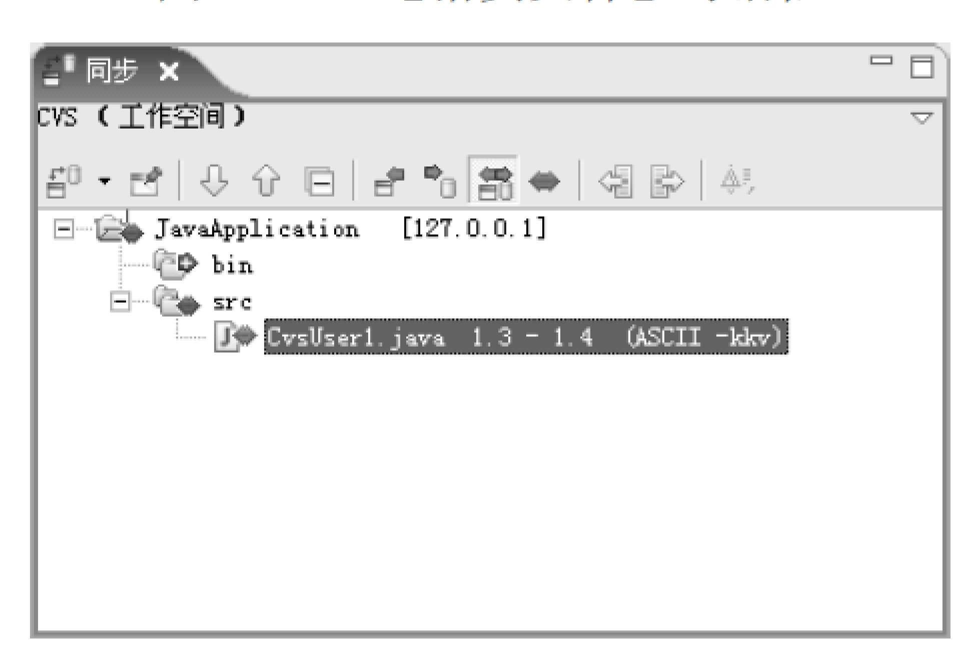


图 11-18　出现冲突

（7）右击同步视图中的“CvsUser1.java”，在快捷菜单中选择【在比较编辑器中打开】命令，可以查看本地跟服务器端的内容差异，如图 11-19 所示。

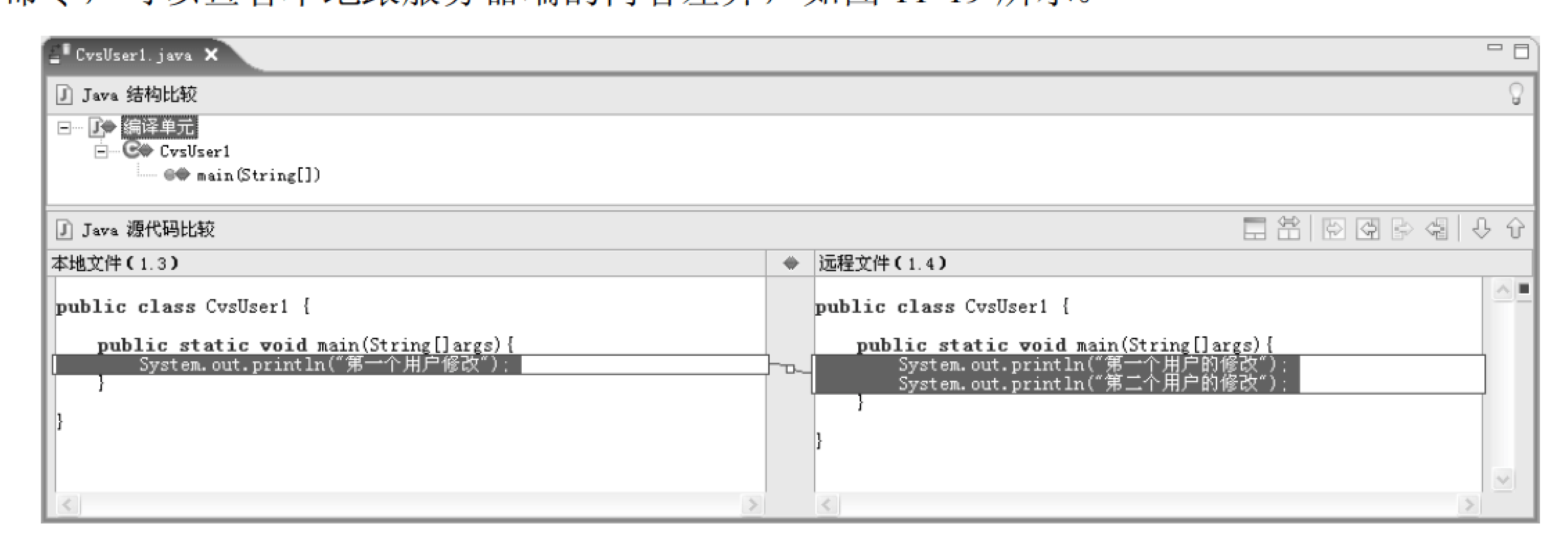


图 11-19　比较编辑器

当出现冲突时就需要由用户来决定采取哪种同步策略：将本地的更新覆盖服务器上的版本还是用服务器上的版本覆盖本地的文件。

（8）如果其他用户修改了 CvsUser1.java 文件，并提交到 CVS 服务器上，更新后服务器上该文件的内容如下所示：

```
public class CvsUser1 {

    public static void main(String[]args){
        System.out.println("第一个用户的修改");
        System.out.println("第二个用户的修改");
    }
}
```

现在本地的版本落后于服务器上的版本，需要更新本地文件。

（9）右击“JavaApplication”工程，在快捷菜单中选择【小组】|【与存储库同步】命令，比较本地和服务器端的内容差异，如图 11-20 所示。



图 11-20　比较编辑器

（10）右击同步视图中的“CvsUser1.java”，在快捷菜单中选择【小组】|【更新】命令，将本地文件更新到与服务器上的版本一致。

11.6　断开项目与 CVS 的连接

当项目开发已经结束，并且团队希望冻结源代码时，可以从 HEAD 资源库删除该项目的最终版本。断开项目与 CVS 的连接将在该项目及其资源上禁用资源库操作，并删除与该项目相关联的 CVS 信息（这一操作是可选的）。可以通过项目上下文菜单中的 Team/Disconnect 选项执行断开连接操作。

跟我做

（1）右击“JavaApplication”工程，在快捷菜单中选择【小组】|【断开连接】命令，打开【确认与 CVS 断开连接】对话框，如图 11-21 所示。



图 11-21　与 CVS 断开连接

（2）单击【是】按钮，JavaApplication 工程断开了与 CVS 服务器的连接，并且从文件系统中删除了 CVS 元信息。

第 3 篇



综合案例

第 12 章　综合实例——光盘资料管理系统

在前述章节分别介绍了 Ant、HSQLDB、Struts、Hibernate 等框架，本章通过一个综合实例光盘资料管理系统来介绍如何将这些优秀的开源框架完美结合，快速开发出一个完整的应用系统。

一个典型的 J2EE 应用基本上都可以分为数据层、逻辑层和表现层 3 个层次，为典型的 MVC 结构。本章的光盘资料管理系统主要综合了 Struts 框架、Hibernate 框架和数据库技术，其为一个典型的应用系统，也是在实际的企业中应用最为广泛的技术架构。可通过本章的学习及在进一步熟悉两个框架的基础上，加深对 MVC 结构的理解。

12.1　需求分析

12.1.1　系统功能分析

光盘资料管理系统可以让拥有大量音乐 CD 的音乐发烧友科学有序地管理其所有 CD 唱片，其作为一个完整的应用系统提供如下功能：

（1）用户权限管理。为了保障系统的应用安全，系统提供安全的用户认证机制，匿名用户无权使用系统的任何功能。包括用户注册、登录、修改密码和注销等功能。

（2）添加音乐 CD 信息功能。通过该管理系统可以方便地添加新的 CD 信息，将 CD 名字、CD 发行公司名称、CD 类型等信息保存到关系数据库中。

（3）音乐 CD 查询、编辑和删除功能。通过 CD 名称的关键字从数据库中查询所有满足条件的 CD 唱片信息，并且分页显示。对查询到的结果可以进行编辑和删除操作。

12.1.2　系统数据流描述

整个系统由用户、6 个动作和 3 张数据表组成，图 12-1 箭头所示为不同部分之间的数据流向。

（1）系统登录：用户登录时将根据用户信息表中的信息验证用户合法性。

（2）用户注销：用户注销时将从系统中退出。

（3）添加 CD 信息：用户填写新音乐 CD 的具体信息，并从 CD 类型表中选择新 CD 所属的类型，将这些信息插入到数据库中。

（4）查询 CD 信息：用户根据关键字从数据库中查询符合条件的音乐 CD 信息。

（5）编辑 CD 信息：用户可以修改查询到的 CD 记录并将其更新到 CD 信息表中。

（6）删除 CD 信息：用户可以将数据库中某条记录删除掉。

图 12-1　系统数据流

12.1.3　数据的存储

光盘资料管理系统的数据库设计是至关重要的，需要保存的系统信息包括用户信息、CD 信息和 CD 类型信息。本小节设计系统的数据库结构，所有表格如表 12-1 所示。

表 12-1　系统的所有表格

表　编　号	表　　名
TBL001	admin
TBL002	CDinfo
TBL003	CDtype

各表详细描述如下所示：

admin 表保存用户的登录信息，userId 为管理员的 ID 编号，userName 为管理员的用户名，userPwd 为管理员的密码。具体表结构如表 12-2 所示。

表 12-2　admin表

名　　称	关　键　字	类　　型	长　　度	值　　域	默　认　值	备　　注
userId	P	自动增长	8	Long		
userName		VC	50			
userPwd		VC	50			

CDinfo 表存储所有 CD 信息，包括 CD 名称、CD 出版公司等信息，其中 cdId 为该 CD 设置的 ID 编码。具体表结构如表 12-3 所示。

表 12-3　CDinfo表

名　　称	关　键　字	类　　型	长　　度	值　　域	默　认　值	备　　注
cdId	P	bigint	8			
cdName		VC	50			
cdCompany		VC	50			
cdAlbum		VC	50			

CDtype 表存储音乐 CD 的类型信息，其中 cdTypeId 是为不同类型配置的 ID 编码。具体表结构如表 12-4 所示。

表 12-4　CDtype表

名　　称	关　键　字	类　　型	长　　度	值　　域	默　认　值	备　　注
cdTypeId	P	int	4			
display		VC	50			

12.1.4　系统所有处理的描述

光盘资料管理系统主要包括对用户管理的操作和对 CD 管理的操作，本小节详细分析系统的业务需求，列出了系统中需要具备的所有操作，如表 12-5 所示。

表 12-5　系统的所有处理

处 理 编 号	处　理　名
1	用户登录
2	用户注销
3	修改密码
4	添加 CD 信息
5	查询 CD 信息
6	编辑 CD 信息
7	删除 CD 信息

用户登录时提供用户名和用户密码，系统验证用户的合法性，并且将验证结果返回给用户。其描述如表 12-6 所示。

表 12-6　用户登录

处　理　名	用 户 登 录
处理编号	1
输入数据流	用户名+用户密码
输出数据流	提示用户登录成功或失败
处理逻辑	根据用户登录信息查询用户信息表，从而验证用户的合法性

用户注销操作是用户从系统中注销，重新返回用户登录页面。其描述如表 12-7 所示。

表 12-7　用户注销

处　理　名	用 户 注 销
处理编号	2
输入数据流	无
输出数据流	用户从系统中注销，重新返回用户登录页面
处理逻辑	用户从系统中注销，重新返回用户登录页面

修改密码操作是用户提供用户名、用户密码、新密码和确认密码等信息，系统验证用户提供的新密码和确认密码是否一致，如果一致就将新密码保存到数据库中。其详细描述如表 12-8 所示。

表 12-8　修改密码

处　理　名	修 改 密 码
处理编号	3
输入数据流	用户名+用户密码+新密码+确认新密码
输出数据流	新密码更新到用户信息表中
处理逻辑	验证用户提供的新密码和确认新密码是否一致，如果一致就将新密码保存到数据库中

添加 CD 信息将用户提交的 CD 名称、CD 公司名称等信息保存到 CD 信息表中。其描述如表 12-9 所示。

表 12-9　添加CD信息

处　理　名	添加 CD 信息
处理编号	4
输入数据流	CD名称+公司名称+CD盒+CD类型
输出数据流	将新的CD信息保存到数据库中
处理逻辑	将用户通过表单提交的CD信息保存到CD信息表中

查询 CD 信息操作根据用户提交的查询关键字从数据库中查询符合条件的 CD 信息。其描述如表 12-10 所示。

表 12-10　查询CD信息

处　理　名	查询 CD 信息
处理编号	5
输入数据流	查询关键字
输出数据流	数据库中满足条件的 CD 信息
处理逻辑	根据用户输入的查询关键字从数据库中查询符合条件的 CD 信息

编辑 CD 信息操作将用户修改的 CD 信息保存到数据库中。其描述如表 12-11 所示。

表 12-11　编辑CD信息

处　理　名	编辑 CD 信息
处理编号	6
输入数据流	CD 名称+公司名称+CD 盒+CD 类型
输出数据流	更新后的新 CD 信息
处理逻辑	将用户更新后的 CD 信息保存到数据库

删除 CD 信息操作将根据用户输入的 CD 名称、公司名称等信息确定某条 CD 记录，并将该条记录从数据库中删除。其描述如表 12-12 所示。

表 12-12　删除CD信息

处　理　名	删除 CD 信息
处理编号	7
输入数据流	CD 名称+公司名称+CD 盒+CD 类型
输出数据流	无
处理逻辑	将该条 CD 信息从数据库中删除

12.2　系统的实现效果

光盘资料管理系统实现对 CD 信息的管理，本节首先介绍系统的运行效果，从而从总体上了解系统的功能，然后将详细介绍系统的具体实现步骤。

跟我做

（1）启动 Tomcat 服务器。

（2）打开 IE 浏览器，在地址栏中输入 http://localhost:8080/cdbox/login.jsp，进入如图 12-2 所示的登录页面。

（3）单击【注册】按钮，打开【新用户注册】窗口，在【用户名】、【用户密码】和【确认密码】文本框中都输入“cd”，如图 12-3 所示。



图 12-2　系统登录页面



图 12-3　新用户注册页面

（4）单击【注册】按钮，完成新用户的注册。以新注册的“cd”用户登录系统，进入“管理控制台”页面，如图 12-4 所示。

（5）单击【添加】超链接，打开【新增音乐 CD】窗口，在【名字】文本框中输入“Debut”，在【公司】文本框中输入“Elektra / Wea”，在【歌手】文本框中输入“Björk”，选择“ROCK”类型，如图 12-5 所示。




图 12-4　管理控制台



图 12-5　添加 CD 信息

（6）单击【添加】按钮，将该 CD 信息添加到 CD 信息表中。

（7）单击【查询/编辑】超链接，打开如图 12-6 所示的【查询 CD】窗口，输入“bu”关键字，单击【查询】按钮，查询结果如图 12-7 所示。



图 12-6　输入查询关键字



图 12-7　查询结果

（8）单击查询结果中编号为 140 记录的【编辑】超链接，打开如图 12-8 所示的【编辑 CD 信息】窗口，修改任意项的内容，单击【更新】按钮，将新内容更新到数据库中。

（9）单击查询结果中编号为 140 记录的【删除】超链接，打开如图 12-9 所示的【删除 CD 信息】窗口，询问用户是否删除该条 CD 信息，单击【确认】超链接，删除该条 CD 信息。



图 12-8　编辑 CD 信息

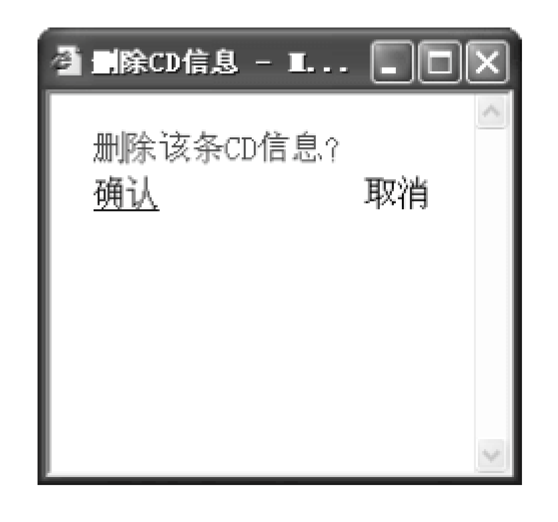


图 12-9　删除 CD 信息

（10）单击【修改密码】超链接，打开如图 12-10 所示的【修改用户密码】窗口，在其中输入“原始密码”、“新密码”和“确认密码”，单击【修改】按钮，完成新密码的修改。

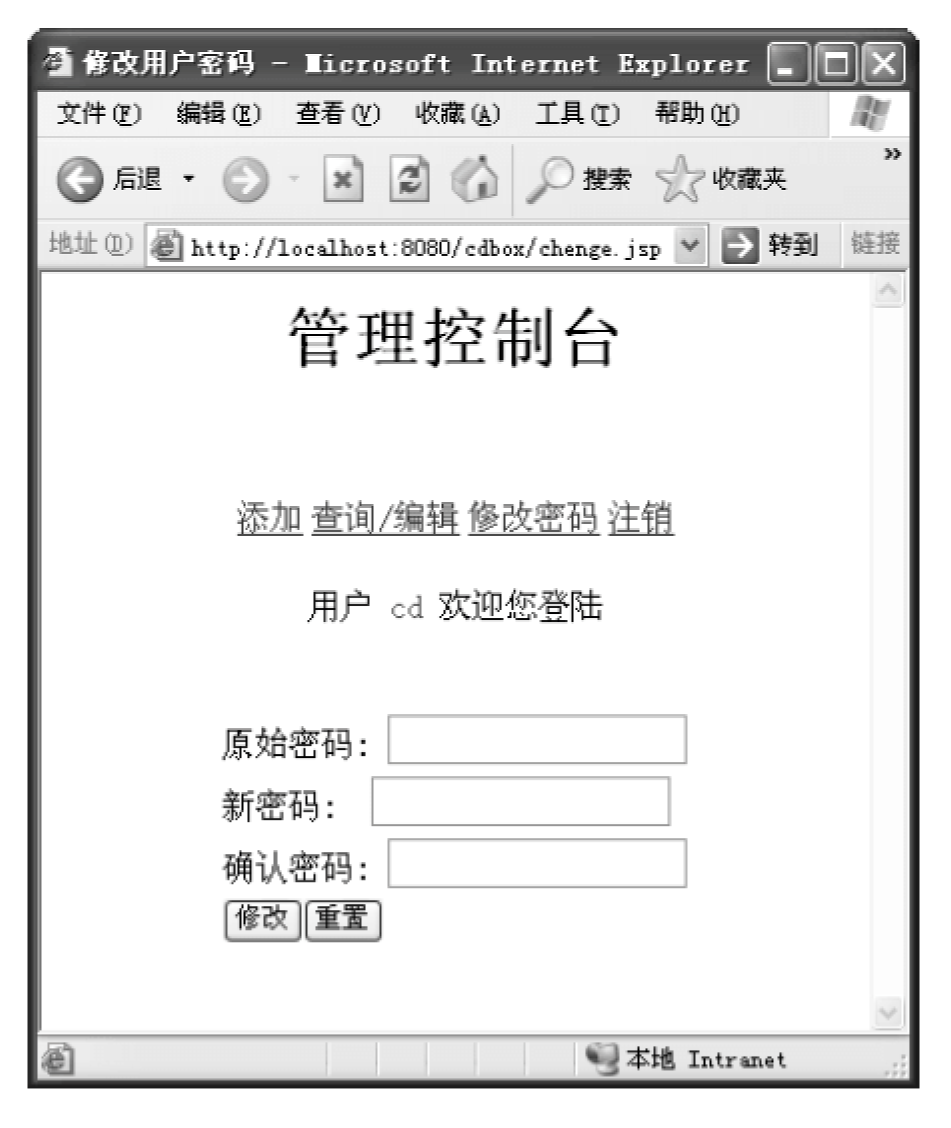


图 12-10　修改用户密码

12.3　配置数据库

CD 管理系统的 CD 信息和用户信息都保存在 SQL Server 数据库中，数据库的设计是系统设计的主要方面。本书介绍数据库的配置。关于 SQL Server 数据库的相关知识可参见其相关文档。

跟我做

（1）打开 SQL Server 企业管理器，创建名称为“MyData”的数据库，如图 12-11 所示。

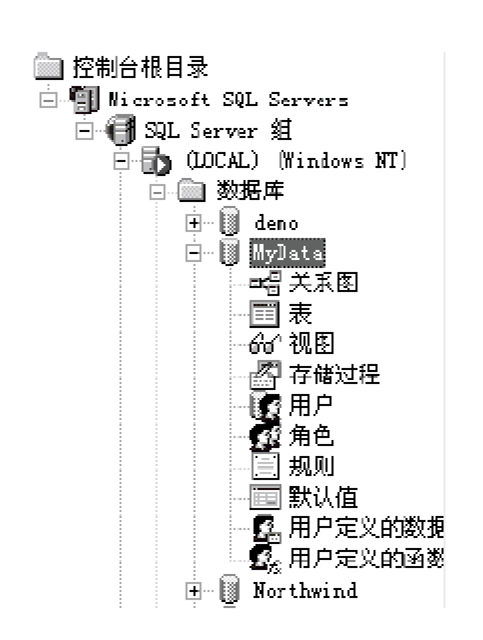


图 12-11　新建 MyData 数据库

（2）打开 SQL Server 的 SQL 查询分析器，选择默认数据库为刚才创建的“MyData”数据库，输入如下 SQL 脚本：

```
//创建 CDinfo 表，保存所有的 CD 信息，以 cdId 为主键
CREATE TABLE [dbo].[CDinfo] (
	[cdName] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
	[cdCompany] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
	[cdAlbum] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
	[cdType] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
	[cdId] [bigint] IDENTITY (1, 1) PRIMARY KEY NOT NULL
)
//创建 CDtype 表，保存所有的 CD 类型，以 cdTypeId 为主键
CREATE TABLE [dbo].[CDtype] (
	[cdTypeId] [int] IDENTITY (1, 1) PRIMARY KEY NOT NULL ,
	[display] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL
)
//创建 admin 表，保存所有的用户信息，以 userId 为主键
CREATE TABLE [dbo].[admin] (
	[userName] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
	[userPwd] [varchar] (50) COLLATE Chinese_PRC_CI_AS NOT NULL ,
	[userId] [bigint] IDENTITY (1, 1) PRIMARY KEY NOT NULL
)
```

（3）单击 ▶ 按钮，执行上述 SQL 脚本，生成 3 个表：CDinfo 表、CDtype 表和 admin 表，分别保存 CD 信息、CD 类型信息和用户信息。

12.4　生成配置文件 hibernate.cfg.xml

Hibernate 运行时需要通过基于 XML 格式文件的方式配置数据库 URL、数据库用户、数据库用户密码、数据库 JDBC 驱动类和数据库 dialect 等信息。本节介绍如何在 Eclipse 中通过 Hibernate Synchronizer 插件快速生成 hibernate.cfg.xml 文件。

hibernate.cfg.xml 文件可以包含构建 SessionFactory 实例的所有配置信息。当使用代码

```
SessionFactory sessions=new Configuration().configure().buildSessionFactory();
```

初始化 Hibernate 时，Hibernate 会在 classpath 中寻找文件名为 hibernate.cfg.xml 的文件。

跟我做

（1）创建名称为“cdbox”的 Java 工程。单击【文件】菜单，选择【新建】|【其他】命令，打开如图 12-12 所示的【新建】对话框。

（2）选择【Hibernate Configuration File】选项，单击【下一步】按钮，在图 12-13 的【输入或选择父文件夹】文本框中选择“cdbox”工程，单击【下一步】按钮，打开数据库配置对话框。



图 12-12　【新建】对话框

图 12-13　选择工程名称

（3）在数据库配置对话框中输入如下数据库配置信息，如图 12-14 所示。

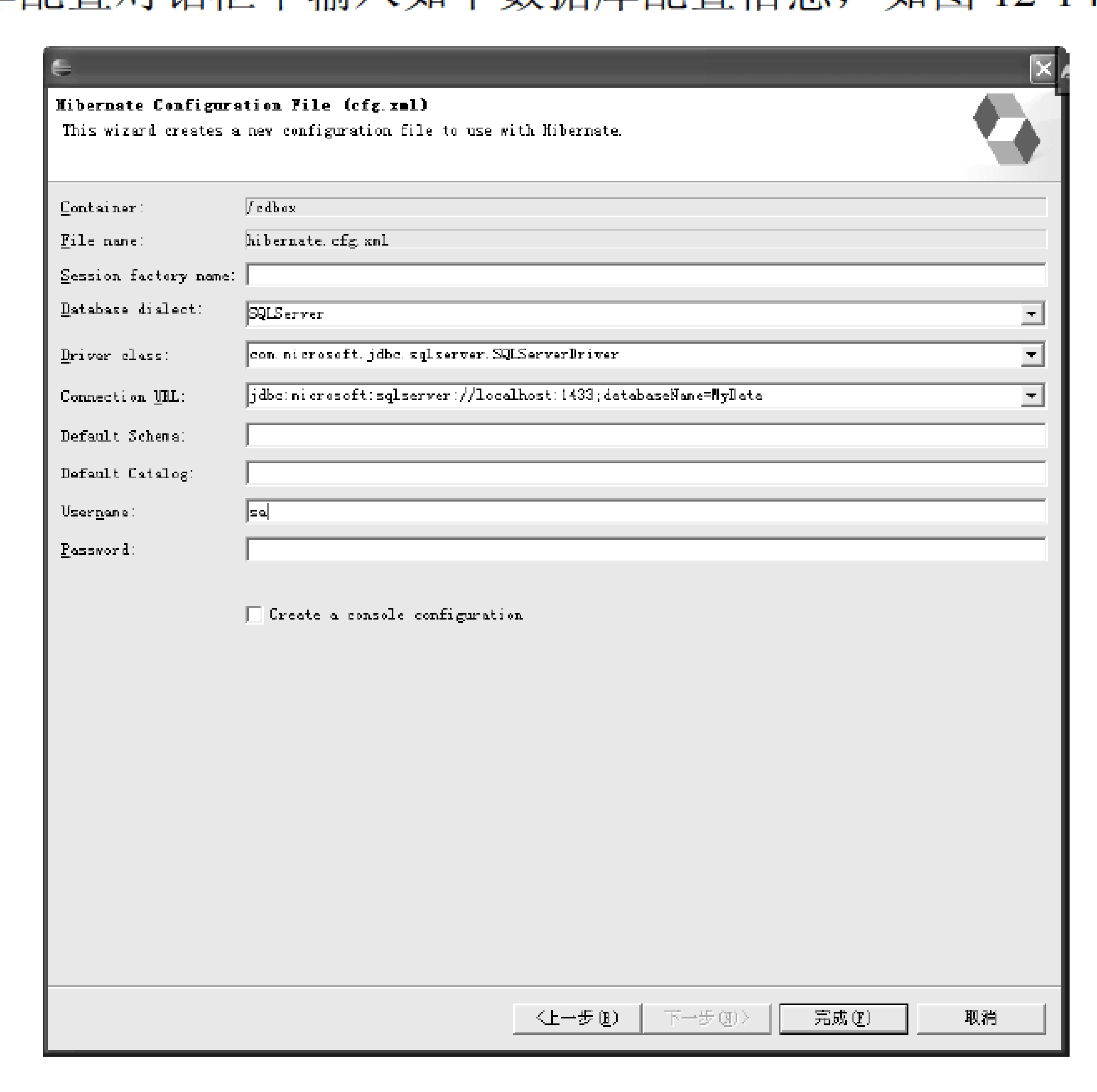


图 12-14　数据库配置对话框

- Database dialect：SQLServer
- Driver Class：com.microsoft.jdbc.sqlserver.SQLServerDriver
- Connection URL：jdbc:microsoft:sqlserver://localhost:1433;databaseName=MyData
- Username：sa（根据实际配置）

（4）单击【完成】按钮，在 cdbox 工程的根目录下生成 hibernate.cfg.xml 文件。其内容如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
          "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory >
    <!--数据库 JDBC 驱动类-->
        <property
name="hibernate.connection.driver_class">com.microsoft.jdbc.sqlserver.SQLServerDriver<
/property>
    <!--数据库密码-->
        <property name="hibernate.connection.password"></property>
     <!--数据库的 URL-->
        <property
name="hibernate.connection.url">jdbc:microsoft:sqlserver://localhost:1433;databaseName=
MyData</property>
     <!--数据库的用户名-->
        <property name="hibernate.connection.username">sa</property>
     <!--每个数据库都有其对应的 Dialet 以匹配其平台特性-->
        <property name="hibernate.dialect">org.hibernate.dialect.SQLServerDialect</property>
    </session-factory>
</hibernate-configuration>
```

12.5　创建持久化对象

Hibernate 最好的使用方法是使用普通的 Java 对象（Plain Old Java Objects，就是 POJOs，有时也称做 Plain Ordinary Java Objects）这种编程模型来进行持久化。一个 POJO 很像 JavaBean，属性通过 getter 和 setter 方法访问，对外隐藏了内部实现的细节。持久化类不需要实现什么特别的接口，也不需要从一个特别的持久化根类继承下来。Hibernate 也不需要使用任何编译器处理，比如字节码增强操作，它独立地使用 Java 反射机制和运行时类增强（通过 CGLIB）。所以，在 Hibernate 中，POJO 的类不需要任何前提条件，即可把它映射成为数据库表。

跟我做

（1）单击 Eclipse 的【窗口】菜单，选择【打开透视图】|【其他】命令，打开【选择透视图】对话框，如图 12-15 所示。选择【Hibernate Console】选项，单击【确定】按钮，

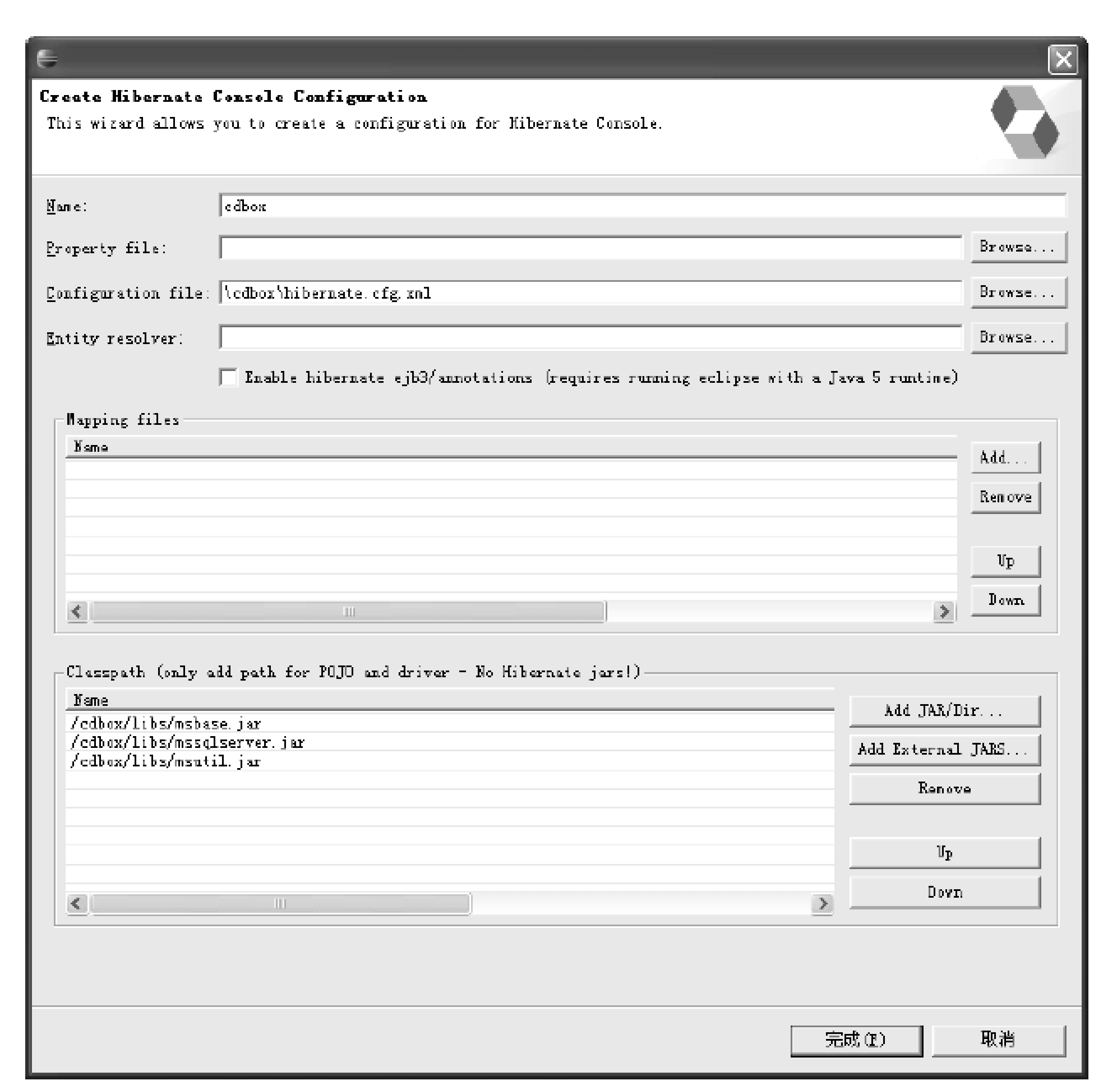
打开【Hibernate Console】透视图。



图 12-15　【选择透视图】对话框

（2）右击 Hibernate Configurations 视图的空白区域，在快捷菜单中选择【Add configuration】命令，出现【Create Hibernate Console Configuration】界面，如图 12-16 所示。

（3）单击【Configuration file】文本框右侧的 Browse... 按钮并选择“\cdbox\ hibernate.cfg.xml”；单击【Classpath】组中的 Add JAR/Dir... 按钮将 SQL Server 数据库的 JDBC 驱动类 msbase.jar、mssqlserver.jar 等加入到 classpath 中；在【Name】文本框中输入“cdbox”，单击【完成】按钮，创建名称为“cdbox”的配置，如图 12-17 所示。



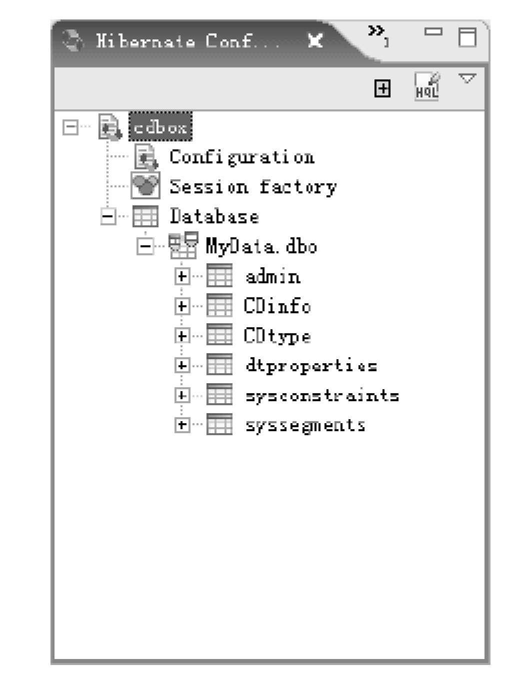


图 12-16　【Create Hibernate Console Configuration】界面　　图 12-17　Hibernate Configuration 视图

（4）单击工具栏中的按钮右边的下拉箭头，从中选择【Hibernate Code Generation】命令，打开【Hibernate Code Generation】对话框。

（5）在【Hibernate Code Generation】对话框中的【Main】选项卡中输入如下配置信息：

- 名称：codegeneration
- Console configuration：cdbox
- Output directory：\cdbox\src
- Package：cdbox.hibernate

创建名称为“codegeneration”的“Hibernate Code Generation”配置，生成的目标代码存放在“\cdbox\src”目录下，其包名为“cdbox.hibernate”。

（6）从【Exporters】选项卡中选中如下复选框，如图 12-18 所示。

- Generate domain code(.java)
- JDK1.5 Constructs(generics,etc)
- Generate DAO code(.java)
- Generate mappings(hbm.xml)

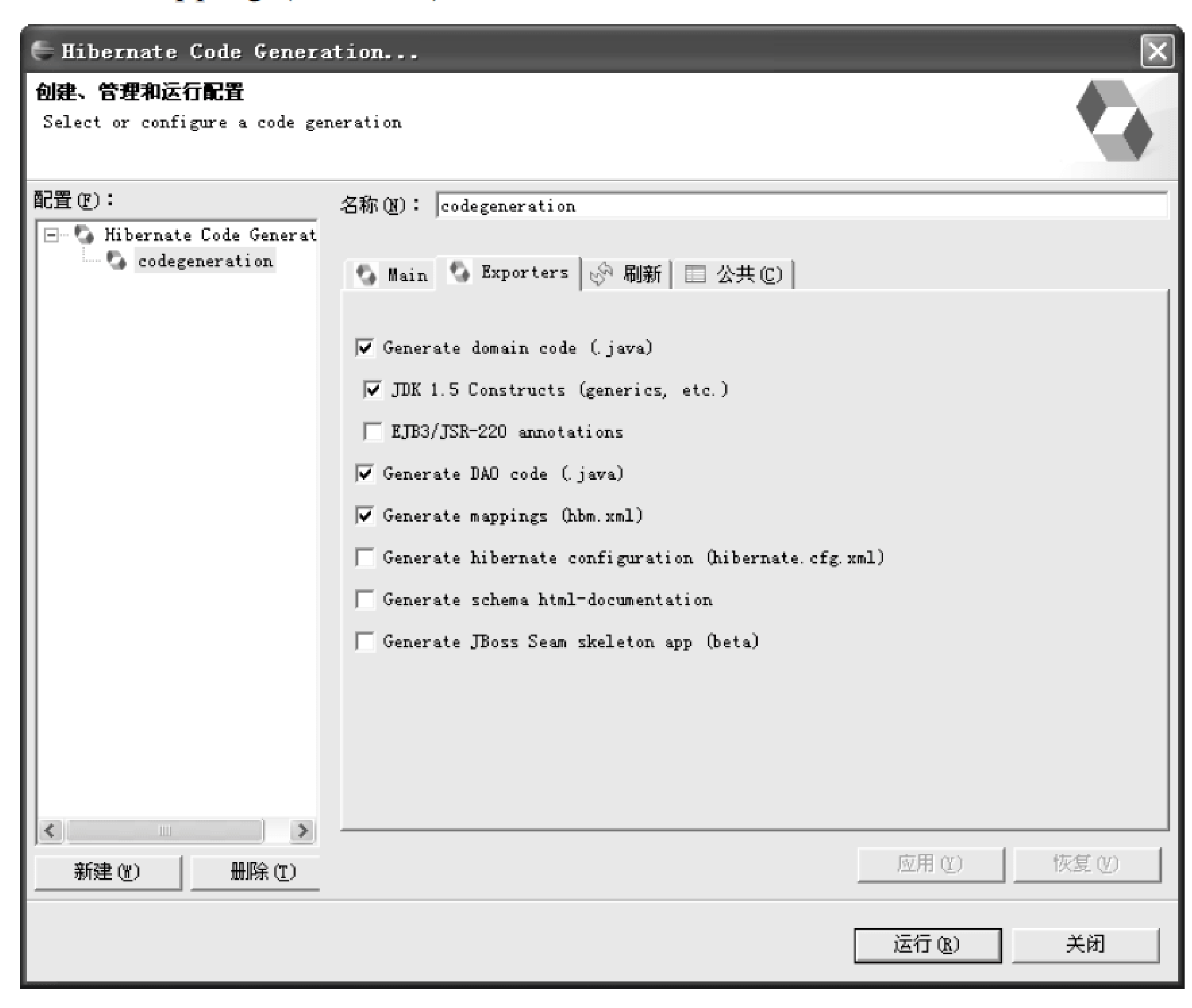


图 12-18　【Exporters】选项卡

（7）单击【运行】按钮，在“cdbox”工程的“cdbox.hibernate”包中生成了几个映射文件。

生成的 CDDate.java 和 User.java 类是标准的 JavaBean，Hibernate 插件根据数据库的表结构自动生成了这两个持久化对象，对于每个属性都有其对应的 getter/setter 方法。

CD.java 文件对应于 MyData 数据库中的 CDinfo 表，包括 CD 名称、公司名称、CD 类型等信息。其内容如下所示：

```
public class CD {
	//CD 名称
	private String cdName;
	//公司名称
	private String cdCompany;
	//CD 夹名称
	private String cdAlbum;
	//CD 类型
	private String cdType;
	//CD 的编号
	private long cdId;

	public CD() {
	}

	/**
	 * 设置 CD 名称属性
	 * @param cdName
	 */
	public void setCdName(String cdName) {
		this.cdName = cdName;
	}

	/**
	 * 设置 CD 公司名称属性
	 * @param cdCompany
	 */
	public void setCdCompany(String cdCompany) {
		this.cdCompany = cdCompany;
	}

	/**
	 * 设置 CD 夹名称属性
	 * @param cdAlbum
	 */
	public void setCdAlbum(String cdAlbum) {
		this.cdAlbum = cdAlbum;
	}

	/**
	 * 设置 CD 类型属性
	 * @param cdType
	 */
	public void setCdType(String cdType) {

		this.cdType = cdType;
	}

	/**
```

```
     * 设置 CDid 属性
     * @param cdId
     */
    public void setCdId(long cdId) {
        this.cdId = cdId;
    }

    /**
     * 取得 CD 名称
     * @return
     */
    public String getCdName() {
        return cdName;
    }

    /**
     * 取得公司名称
     * @return
     */
    public String getCdCompany() {
        return cdCompany;
    }

    /**
     * 取得 CD 夹名称属性
     * @return
     */
    public String getCdAlbum() {
        return cdAlbum;
    }

    /**
     * 取得 CD 类型属性
     * @return
     */
    public String getCdType() {

        return cdType;
    }

    /**
     * 取得 CDID 属性
     * @return
     */
    public long getCdId() {
        return cdId;
    }
}
```

User.java 类对应于 MyData 数据库中的 admin 表，包括用户名、用户密码和用户 ID 属

性。其内容如下所示：

```
public class User   {
	// 用户名
	private String userName;

	// 用户密码
	private String userPwd;

	// 用户 ID
	private long userId;

	/**
	 * 构造函数
	 */
	public User() {
	}

	/**
	 * 设置用户名属性
	 *
	 * @param userName
	 */
	public void setUserName(String userName) {
		this.userName = userName;
	}

	/**
	 * 设置用户密码
	 *
	 * @param userPwd
	 */
	public void setUserPwd(String userPwd) {
		this.userPwd = userPwd;
	}

	/**
	 * 设置用户 ID
	 *
	 * @param userId
	 */
	public void setUserId(long userId) {
		this.userId = userId;
	}

	/**
	 * 取得用户名属性
	 *
	 * @return
	 */
```

```
        public String getUserName() {
                return userName;
        }

        /**
         * 取得用户密码属性
         *
         * @return
         */
        public String getUserPwd() {
                return userPwd;
        }

        /**
         * 取得用户 ID 属性
         *
         * @return
         */
        public long getUserId() {
                return userId;
        }
}
```

CD.hbm.xml 描述了 CD 类和 CDinfo 表之间的映射关系，其内容如下所示：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping >
       <!--CD 类和 CDinfo 表对应关系描述-->
       <class name="CD" table="CDinfo">
              <!--cdid 属性对应 cdid 字段-->
              <id name="cdId" column="cdId">
                     <generator class="identity"/>
              </id>
              <!--cdName 属性对应 cdName 字段，并且非空-->
              <property name="cdName" column="cdName" not-null="true"/>
              <!--cdCompany 属性对应 cdCompany 字段，并且非空-->
              <property name="cdCompany" column="cdCompany" not-null="true"/>
              <!--cdAlbum 属性对应 cdAlbum 字段，并且非空-->
              <property name="cdAlbum" column="cdAlbum" not-null="true"/>
              <!--cdType 属性对应 cdType 字段，并且非空-->
              <property name="cdType" column="cdType" not-null="true"/>

       </class>
</hibernate-mapping>
```

User.hbm.xml 文件描述了 User 类和 admin 表之间的映射关系，其内容如下所示：

```
<?xml version="1.0"?>
```

```
<!DOCTYPE hibernate-mapping
PUBLIC "-//Hibernate/Hibernate Mapping DTD 2.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping >
        <!--User 类和 admin 表对应关系描述-->
    <class name="User" table="admin">
        <!--userid 属性对应 userid 字段-->
        <id name="userId" column="userId" type="long">
            <generator class="identity"/>
        </id>
         <!--userName 属性对应 userName 字段，并且非空-->
        <property name="userName" column="userName" type="string" not-null="true"/>
         <!--userPwd 属性对应 userPwd 字段，并且非空-->
        <property name="userPwd" column="userPwd" type="string" not-null="true"/>

    </class>
</hibernate-mapping>
```

12.6　对数据库操作的封装

通过 Hibernate 可以简化对数据库的操作，本节首先创建一个 DBManager 类，用于管理 session，然后将 CD 管理系统中涉及的对数据库操作都封装到该类中。

12.6.1　创建 DBManager 类

DBManager 根据 Hibernate 配置文件创建了 SessionFactory，同时创建了 Session 实例和 Transaction 实例，并且为 Session 实现了打开和关闭操作。SessionFactory 是线程安全的，很多线程可以同时访问它，获取 Session。Session 不是线程安全的，它代表与数据库之间的一次操作。在 Session 中，每个数据库操作都是在一个事务（Transaction）中进行的，这样就可以隔离开不同的操作（甚至包括只读操作）。用户使用 Hibernate 的 Transaction API 来从底层的事务策略中（本例中是 JDBC 事务）脱身。这样，如果需要把程序部署到一个由容器管理事务的环境中去（使用 JTA），就不需要更改源代码。本节创建 DBManager 类，用于管理对数据库的所有操作。

跟我做

在“cdbox”工程的“src”文件夹中创建“db”包，并在该包中创建“DBManager.java”文件，编辑该类，输入如下内容：

```
public class DBManager {

    // 生成 Session 的工厂类
    private SessionFactory sessionFactory = null;
```

```
    // Session 类
    private Session session = null;

    // 事务类
    private Transaction transaction = null;

    // Query 类属性
    private Query query = null;

    // 页数
    public static int PAGECOUNT;

    /**
     * DBManager 的构造函数
     */
    public DBManager() {
        // 根据 Hibernate 配置文件创建 SessionFactory
        sessionFactory = new Configuration().configure().buildSessionFactory();
        // 创建 Session 类
        session = sessionFactory.openSession();
        // 创建事务
        transaction = session.beginTransaction();
    }

    /**
     * 打开 Session 类
     */
    public void openSession() {
        session = sessionFactory.openSession();
    }
    /**
     * 关闭 Session
     */
    public void close() {
        session.close();
    }
}
```

12.6.2　创建用户操作方法

Hibernate 有不同的方法从数据库中取回对象。最灵活的方式是使用 Hibernate 查询语言（HQL），这是一种容易学习的语言，是对 SQL 的面向对象的强大扩展。Hibernate 也提供一种面向对象的按条件查询 API，可以执行公式化的类型安全的查询。当然，Hibernate 在所有与数据库的交互中都使用 PrepatedStatement 和参数绑定。用户也可以使用 Hibernate 的直接 SQL 查询功能，或者在特殊情况下从 Session 获取一个原始的 JDBC 连接。本小节将在 DBManager 类中加入对用户的操作，包括验证用户的合法性、添加用户、更新用户信息等。

跟我做

（1）在 DBManager 类中实现 checkUser 方法，验证用户是否为合法用户。其方法体如下所示：

```
/**
	 * 验证用户是否为合法用户
	 *
	 * @param user
	 * @return
	 */
	public boolean checkUser(User user) {
		// 是否为合法用户标记
		boolean flag = false;
		try {
			// 根据参数中的用户名和密码从数据库中查询是否存在该用户，从而验证给用户的合
法性
			query = session.createQuery("from User user where user.userName='"
					+ user.getUserName() + "' and userPwd='"
					+ user.getUserPwd() + "'");
			// 查询到的结果
			List list = query.list();
			// 如果查询到的结果大于 0，说明该用户为合法用户，将标记 flag 置为 true
			if (list.size() > 0) {
				flag = true;
			} else {
				// 如果没有查询到合法用户，就说明该用户为非法用户
				flag = false;
			}
		} catch (Exception e) {
			flag = false;
			e.printStackTrace();
		}
		return flag;
	}
```

该方法根据用户的登录信息从数据库中查询，如果返回的结果不为空，说明该用户为合法用户，登录成功，否则提示用户登录失败。

（2）在 DBManager 类中增加 updateUserPwd 方法，该方法将修改用户的密码，将新的用户密码保存到数据库中。其方法体的内容如下所示：

```
	/**
	 * 更新给定用户的密码
	 *
	 * @param user
	 * @return
	 */
	public boolean updateUserPwd(User user) {
```

```
        // 通过 Session 查询给定用户名的用户
        query = session.createQuery("from User user where user.userName='"
                + user.getUserName() + "'");
        // 从查询结果中取得第一个值
        User firstUser = (User) query.list().get(0);
        // 设置用户的密码
        firstUser.setUserPwd(user.getUserPwd());
        // 将更新后的信息同步到数据库中
        session.update(firstUser);
        // 提交事务
        transaction.commit();
        return transaction.wasCommitted();
    }
```

该方法通过 Session 执行了查询操作。

（3）在 DBManager 类中增加 hasUser 方法，该方法判断数据库中是否存在特定的用户。其方法体如下所示：

```
/**
     * 判定某个用户是否为合法用户
     *
     * @param user
     * @return
     */
    private boolean hasUser(User user) {
        boolean flag = false;
        // 从数据库中查询是否存在给定用户名的用户
        query = session.createQuery("from User user where user.userName='"
                + user.getUserName() + "'");
        // 如果查询结果不为 0 表示该用户为合法用户
        if (query.list().size() != 0) {
            flag = true;
        }
        return flag;
    }
```

（4）在 DBManager 类中增加 addUser 方法，该方法将往数据库中添加新的用户信息。其方法体如下所示：

```
    /**
     * 添加新用户
     *
     * @param user
     * @return
     */
    public boolean addUser(User user) {
        // 判定是否已经存在该用户
        if (hasUser(user)) {
            return false;
        }
```

```
            // 通过 Session 将新的用户信息保存到数据库中
            session.save(user);
            // 提交事务
            transaction.commit();
            return transaction.wasCommitted();
        }
```

12.6.3　创建 CD 操作方法

本小节创建关于 CD 信息的操作方法，包括搜索 CD、添加 CD 和删除 CD 等操作。通过

```
session.createQuery("from CDDate cd where cd.cdName like '%"+ value + "%'");
session.update(cd);
session.delete(cd);
```

分别实现对 CD 信息的搜索、添加和删除操作。

跟我做

（1）在 DBManager 类中实现 searchCD 类，根据查询关键字从数据库中查询满足条件的 CD 信息。其方法体如下所示：

```
/**
     * 根据关键字从数据库中查询满足条件的记录
     *
     * @param value
     * @param page
     * @param count
     * @return
     */
    public List searchCD(String value, int page, int count) {
        List list = null;
        // 最后一页的页码
        int pagelast = 0;
        try {
            // 从数据库中查询 CD 名称满足某个条件的所有 CD 信息，并且分页显示出来
            query = session
                    .createQuery("from CDDate cd where cd.cdName like '%"
                            + value + "%'");
            // 根据查询到的记录数和每页显示的记录数决定当前页显示的内容和页数
            if (query.list().size() / count == 0) {
                PAGECOUNT = query.list().size() / count;
            } else {
                PAGECOUNT = query.list().size() / count + 1;
                pagelast = query.list().size() / count;
            }
            int begin = page * count - count;
```

```
                int end = page * count;
                if (page == PAGECOUNT) {
                    end = query.list().size();
                }
                list = query.list().subList(begin, end);
            } catch (Exception ex) {
                list = null;
                ex.printStackTrace();
            }
            return list;
        }
```

该方法根据用户输入的关键字从数据库中查询满足条件的记录，并且将查询到的结果分页显示。

（2）在 DBManager 类中增加 updateCD 方法，该方法用来将修改后的 CD 信息保存到数据库中。其方法体如下所示：

```
/**
     * 将修改后的 CD 信息更新到数据库中
     *
     * @param cd
     * @return
     */
    public boolean updataCD(CD cd) {
        // 通过 session 将 CD 信息更新到数据库中
        session.update(cd);
        // 提交事务
        transaction.commit();
        return transaction.wasCommitted();
    }
```

该方法通过 Session 类的 update 方法将修改后的 CD 信息保存到数据库中。

（3）在 DBManager 类中增加 getCD 方法，该方法根据 CD 的 id 从数据库中查询到其详细信息。其方法体如下所示：

```
    /**
     * 根据 id 信息从数据库中找到对应的信息
     *
     * @param id
     * @return
     */
    public CD getCD(long id) {
        // 通过 Session 从数据库中取得 CD 信息
        CD cd = (CD) session.load(CD.class, id);
        // 提交事务
        transaction.commit();
        return cd;
    }
```

该方法通过 Session 的 load 方法从数据库中取得 CD 类。

（4）在 DBManager 类中增加 deleteCD 方法，该方法将给定的 CD 信息从数据库中删除。其方法体如下所示：

```
/**
 * 从数据库中删除给定 id 的 CD 信息
 *
 * @param id
 * @return
 */
public boolean deleteCD(long id) {
        // 通过 Session 取得给定 id 的 CD 信息
        CD cd = (CD) session.get(CD.class, id);
        // 将该条 CD 信息从数据库中删除
        session.delete(cd);
        // 提交事务
        transaction.commit();
        return transaction.wasCommitted();
}
```

该方法通过 Session 的 get 方法取得给定的 CD 信息。

（5）在 DBManager 类中增加 addCD 方法，该方法将往数据库中添加新的 CD 信息。其方法体如下所示：

```
/**
 * 添加新的 CD 信息
 *
 * @param cd
 * @return
 */
public boolean addCD(CD cd) {
        // 通过 Session 将给定 CD 的信息保存到数据库中
        session.save(cd);
        // 提交事务
        transaction.commit();
        return transaction.wasCommitted();
}
```

该方法通过 Session 的 save 方法将 CD 信息保存到了数据库中。

12.7　使用 JSP 实现视图层

CD 管理系统用 Struts 框架实现了视图层和逻辑控制层，其视图部分以 JSP 作为实现手段，接受用户的输入并将结果反馈给用户。

12.7.1　创建用户登录页面

用户登录页面通过 userName 和 userPwd 文本框收集用户名和用户密码，并将这些信息提交给 loginAction，处理用户的身份验证。

其中的<form name="form1" method="post" action="loginAction.do">语句表明与 login.jsp 页面相对应的 loginAction。

跟我做

（1）在“cdbox”工程中创建“pages”文件夹，在新创建的“pages”文件夹下创建“login.jsp”文件。

（2）编辑 login.jsp 文件，在文件中输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<html>
<head>
<title>音乐 CD 管理系统</title>
<script type="javaScript">
<!--提交表单-->
function submitForm()
{
      <!--用户名不能为空-->
     if(document.form1.userName.value=="")
         {
            alert("请输入用户名");
            document.form1.userName.focus();
            return false;
         }else if(document.form1.userPwd.value=="")
         {
            <!--密码不能为空-->
            alert("请输入密码");
            document.form1.userPwd.focus();
            return false;
         }else
         {
           return true;
         }
}
</script>
</head>
<body bgcolor="#ffffff">
<center>
  <h1>音乐 CD 管理系统</h1>
<!--用于创建 HTML 表单，它能够将 HTML 表单的字段和 ActionForm Bean 的属性关联起来-->
  <form name="form1" method="post" action="loginAction.do">
  <br>
  <br>
```

```
    <table align="center">
      <tr>
        <td>用户名</td>
        <td>
        <!--输入用户名文本框-->
          <input type="text" name="userName"/>
        </td>
      </tr>
      <tr>
        <td>密码</td>
        <td>
          <!--输入用户密码文本框-->
          <input type="password" name="userPwd"/>
        </td>
      </tr>
      <tr>
        <td>      </td>
        <td>
          <!--登录按钮-->
          <input type="submit" onclick="submitForm()" name="Submit" value="登录">
            
          <!--重置按钮-->
          <input type="reset" value="重置">
            
          <!--注册按钮，单击后转到 reg.jsp 页面-->
          <input type="button" value="注册" onclick="javascript:window.open('reg.jsp','jszx', 'width=
200, height=300,toolbar=no, status=no, menubar=no, resizable=yes, scrollbars=yes');"/>
        </td>
      </tr>
    </table>
    <br/>
    </form>
</center>
</body>
</html>
```

12.7.2　创建用户注册页面

用户注册页面完成新用户信息的收集并将其提交给相应的 Action 来处理，本小节创建用户注册页面。

其中的<form action="regAction.do">语句表明与 reg.jsp 页面相对应的 loginAction。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“reg.jsp”文件。

（2）编辑 reg.jsp 文件，输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<!--声明和加载 Struts 标签库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<html>
<head>
<title>新用户注册</title>
<script type="javascript">
function check()
{
  <!--新用户注册必须输入用户名-->
  if(document.form1.userName.value=="")
  {
    alert("请输入用户名");
    document.form1.userNamefocus();
    return false;
  }else if(document.form1.userPwd.value=="")
  {
    <!--新用户注册必须输入用户密码-->
    alert("请输入密码");
    document.form1.userPwd.focus();
    return false;
  }
  else if(document.form1.reigthUserPwd.value=="")
  {
    <!--新用户注册必须输入用户确认密码-->
    alert("请输入确认密码");
    document.form1.reigthUserPwd.focus();
    return false;
  }else if(document.form1.userPwd.value!=document.form1.reigthUserPwd.value)
  {
    <!--判断用户密码和确认密码是否一致-->
    alert("新密码和确认密码不一致!");
    return false;
  }else
  {
    return true;
  }
}
</script>
</head>
<body bgcolor="#ffffff">
<center>
  <!--将用户信息提交给 regAction 来处理-->
  <form action="regAction.do">
  <c:if test="${empty requestScope.regresult}">
    <table>
      <tr>
        <td>            用户名:
            
```

```
            <!--输入用户名文本框-->
            <input type="text" name="userName"/>
          </td>
        </tr>
        <tr>
          <td>           用户密码:
              
            <!--输入用户密码文本框-->
            <input type="password" name="userPwd"/>
          </td>
        </tr>
        <tr>
          <td>           确认密码:
             
            <!--输入确认密码文本框-->
            <input type="password" name="reigthUserPwd"/>
          </td>
        </tr>
        <tr>
          <td>
            <!--注册按钮-->
            <input type="submit" onclick="check()" value="注册"/>
            <input type="reset" value="重置"/>
          </td>
        </tr>
      </table>
    </c:if>
     <!--判断用户注册是否成功-->
    <c:if test="${requestScope.regresult==true}">
      <table>
        <tr>
          <td>
            <!--提示用户注册成功-->
            <font color="red">恭喜注册成功!</font>
          </td>
        </tr>
        <tr>
          <td>
            <!--单击“现在登录”链接至 login.jsp 页面-->
            <a  href="javascript:opener.location.href='login.jsp';javascript:window.close(this);">现在
登录</a>
          </td>
        </tr>
      </table>
    </c:if>
    <!--如果用户登录失败-->
    <c:if test="${requestScope.regresult==false}">
      <table>
        <tr>
```

```
            <td>
              <!--提示用户注册失败-->
              <font color="red">注册失败！可能用户名已存在或者含有非法字符！请再尝试</font>
            </td>
          </tr>
          <tr>
            <td>
              <!--单击“继续注册”链接至 reg.jsp 页面中-->
              <a href="reg.jsp">继续注册</a>
            </td>
            <td>
              <!--单击“放弃”放弃登录-->
              <a onclick="javascript:window.close(this);">放弃</a>
            </td>
          </tr>
        </table>
      </c:if>
    </form>
</center>
</body>
</html>
```

12.7.3　创建系统控制台页面

index.jsp 页面是整个系统的控制台，系统的所有功能都能在这里找到其相应的链接。分别链接到 addCD.jsp、selectCD.jsp、change.jsp 等页面。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“index.jsp”文件。

（2）编辑 index.jsp 文件，输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<!--声明和加载 Struts 标签库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>音乐 CD 管理系统</title>
</head>
<body bgcolor="#ffffff">
<center>
    <h1>管理控制台</h1>
    <br>
    <br>
    <table>
      <tr>
        <td>
```

```
        <!--单击“添加”将转移到 add.jsp 页面中-->
          <a href="addCD.jsp">添加</a>
        </td>
        <td>
        <!--单击“查询/编辑”将转移到 select.jsp 页面中-->
          <a href="selectCD.jsp">查询/编辑</a>
        </td>
        <td>
          <!--单击“修改密码”将转移到 chenge.jsp 页面中-->
          <a href="change.jsp">修改密码</a>
        </td>
        <td>
        <!--单击“注销”转移到 outAction 中-->
          <a href="outAction.do">注销</a>
        </td>
      </tr>
    </table>
</center>
</body>
</html>
```

12.7.4　创建新增 CD 信息页面

新增 CD 信息页面收集用户输入的 CD 名称、公司名称、歌手和唱片类型等信息（其中唱片信息从数据库中读取），并将这些信息提交到数据库中，将结果信息返回给用户。

其中的<form name="form1" action="addAction.do" method="POST">语句表明与 add.jsp 页面相对应的 addAction。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“add.jsp”文件。

（2）编辑 add.jsp 文件，输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<!--声明和加载 Struts 标签库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>新增音乐 CD</title>
<script type="javascript">
<!--表格提交函数，判断用户输入是否输入足够信息-->
function submitForm()
{
  if(document.form1.cdName.value=="")
  {
   <!--必须输入 CD 名称信息-->
```

```
        alert("请输入 CD 名称");
        document.form1.cdName.focus();
        return false;
    }else if(document.form1.cdAlbum.value=="")
    {
        <!--必须输入 CD 盒信息-->
        alert("请输入 CD 盒信息");
        document.form1.cdAlbum.focus();
        return false;
    }else if(document.form1.cdCompany.value=="")
    {
        <!--必须输入公司名称信息-->
        alert("请输入公司名称信息");
        document.form1.cdCompany.focus();
        return false;
    }
    else
    {
      return true;
    }
}
</script>
</head>
<body bgcolor="#ffffff">
<!--包含 index.jsp 页面-->
<jsp:include flush="false" page="index.jsp"/>
<center>
    <!--该表单提交的内容将交给 addAction 来处理-->
    <form name="form1" action="addAction.do" method="POST">
    <c:if test="${empty requestScope.addresult}">
        <table>
            <tr>
                <td>                名字
                     
                    <!--输入“名字”的文本框-->
                    <input type="text" name="cdName"/>
                </td>
            </tr>
            <tr>
                <td>                公司
                     
                    <!--输入“公司”的文本框-->
                    <input type="text" name="cdCompany"/>
                </td>
            </tr>
            <tr>
                <td>                歌手
                     
                    <!--输入“歌手”的文本框-->
```

```
                <input type="text" name="cdAlbum"/>
            </td>
        </tr>
        <tr>
            <td>                类型
                 
                <!--从关系数据库中读取 CD 类型信息,数据库的连接信息-->
                <sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver" url="jdbc:
microsoft:sqlserver://127.0.0.1:1433;databasename=MyData" user="sa" password="" var="db"
scope="request"/>
                <!--数据库查询的 SQL 语句-->
                <sql:query var="query" dataSource="${db}">select * from CDtype</sql:query>
                <select name="cdType">
                    <!--将查询到的结果集显示到控件中-->
                    <c:forEach var="type" items="${query.rows}">
                        <option value="${type.display}">${type.display}                </option>
                    </c:forEach>
                </select>
            </td>
        </tr>
        <tr>
            <td>
                <!--“添加”按钮-->
                <input type="submit" onclick="submitForm()" value="添加"/>
                 <!--“清空”按钮-->
                <input type="reset" value="清空"/>
            </td>
        </tr>
    </table>
  </c:if>
  <c:if test="${requestScope.addresult==true}">
  <table>
  <tr>
  <td><font color="red">恭喜!!!添加成功</font></td>
  </tr>
  <tr>
  <td><a href="add.jsp">继续添加</a></td>
  <td><a href="index.jsp">回到主页</a></td>
  </tr>
  </table>
  </c:if>
  <c:if test="${requestScope.addresult==false}">
   <table>
  <tr>
  <td><font color="red">添加失败!!</font></td>
  </tr>
  <tr>
  <td><a href="add.jsp">重新添加</a></td>
  <td><a href="index.jsp">回到主页</a></td>
```

```
  </tr>
  </table>
  </c:if>
  </form>
</center>
</body>
</html>
```

12.7.5　创建查询 CD 信息页面

查询 CD 信息页面根据用户输入的关键字从数据库中查询满足条件的 CD 信息并且以分页的表格形式显示给用户，并且提供导航条，用户可以快速地切换到给定页面。

其中的<form name="form1" action="selectAction.do" method="POST">语句表明与 search.jsp 页面相对应的 addAction。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“search.jsp”文件。

（2）编辑 search.jsp 文件，输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<!--声明和加载 Struts 标签库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>查询 CD</title>
<script type="javaScript">
<!--验证用户是否输入了关键字信息-->
function submitForm()
{
     if(document.form1.selectValue.value=="")
         {
             <!--提示用户应该输入“关键字”信息-->
             alert("请输入查找关键字");
             document.form1.selectValue.focus();
             return false;
         }else
         {
            return true;
         }
}
<!--要跳转的页数-->
function toPage()
{
  if(document.form1.pageText.value=="")
  {
```

```
        <!--提示用户要前往的页数-->
              alert("请输入要前往的页数");
             document.form1.pageText.focus();
             return false;
  }else
        {
            a=document.form1.pageText.value;
            if(a<=0||a>=${requestScope.pagecount})
               a=${requestScope.page}
            document.form1.action = "selectAction.do?page="+a+"&selectValue=${requestScope.
selectValue}";
            return true;
        }
}
</script>
</head>
<body bgcolor="#ffffff">
<!--包含 index.jsp 页面-->
<jsp:include flush="false" page="index.jsp"/>
<center>
  <!--该页面输入的信息将提交给 selectionAction 来处理-->
  <form name="form1" action="selectAction.do" method="POST">
  <table>
    <tr>
      <td>输入查询的关键字</td>
      <td>
        <!--输入关键字的文本框-->
        <input type="text" name="selectValue" value="${requestScope.selectValue}"/>
      </td>
      <td>
        <!--“查询”按钮-->
        <input type="submit" onclick="submitForm()" value="查询"/>
      </td>
    </tr>
  </table>
  <c:if test="${not empty sessionScope.selectList}">
    <!--以表格形式显示查询结果-->
    <table border="1" cellpadding="3" cellspacing="3">
     <!--表格的表头行-->
      <tr>
        <th>编号</th>
        <th>名字</th>
        <th>公司</th>
        <th>歌手</th>
        <th>类型</th>
        <th>操作</th>
      </tr>
      <c:forEach var="cddate" items="${sessionScope.selectList}">
        <tr>
```

```
                <td>${cddate.cdId}              </td>
                <td>${cddate.cdName}               </td>
                <td>${cddate.cdCompany}              </td>
                <td>${cddate.cdAlbum}              </td>
                <td>${cddate.cdType}              </td>
                <td>
                <!--对该条记录的“编辑”动作-->
                  <a
onclick="javascript:window.open('editAction.do?id=${cddate.cdId}','jszx','width=650,height=500,
toolbar=no, status=no, menubar=no, resizable=yes, scrollbars=yes');">编辑</a>
                   
                 <!--对该条记录的“删除”动作-->
                  <a
onclick="javascript:window.open('del.jsp?id=${cddate.cdId}','jszx','width=200,height=300,toolbar=
no, status=no, menubar=no, resizable=yes, scrollbars=yes');">删除</a>
                </td>
              </tr>
            </c:forEach>
          </table>
          <!--显示页码表格-->
          <table>
            <tr>
              <td>
                <a href="selectAction.do?action=frist&selectValue=${requestScope.selectValue}">首页
</a>
              </td>
              <td>
                <c:if test="${requestScope.page==1}">上一页</c:if>
                <c:if test="${requestScope.page!=1}">
                  <a
href="selectAction.do?action=next&page=${requestScope.page-1}&selectValue=${requestScope.
selectValue}">上一页</a>
                </c:if>
              </td>
              <td>
                <c:if test="${requestScope.page==requestScope.pagecount}">下一页</c:if>
                <c:if test="${requestScope.page!=requestScope.pagecount}">
                  <a
href="selectAction.do?action=next&page=${requestScope.page+1}&selectValue=${requestScope.
selectValue}">下一页</a>
                </c:if>
              </td>
              <td>
                <a href="selectAction.do?action=last&selectValue=${requestScope.selectValue}">尾页</a>
              </td>
              <td>${requestScope.page}              /
${requestScope.pagecount}           </td>
              <td>            转到
                <input type="text" size="2" name="pageText" onkeyup="value=value.replace(/[^\d]/g,'')
```

```
"onbeforepaste="clipboardData.setData('text',clipboardData.getData('text').replace(/[^\d]/g,''))"
value="${requestScope.page}"/>
                <input type="submit" onclick="toPage()" value="GO"/>
            </td>
        </tr>
    </table>
  </c:if>
  </form>
</center>
</body>
</html>
```

12.7.6　创建修改用户密码页面

修改用户密码页面提示用户输入原密码、新密码和确认密码，然后将这些信息提交到数据库中，并将结果返回给用户。

其中的<form name="form1" action="changeAction.do" method="POST">语句表明与 change.jsp 页面相对应的 changeAction。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“change.jsp”文件。

（2）编辑 change.jsp 文件，输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<!--声明和加载 Struts 标签库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>修改用户密码</title>
<script type="javascript">
<!--检查用户是否输入了足够的信息-->
function check()
{
  if(document.form1.oldUserPwd.value=="")
  {
    <!--提示用户输入原始密码-->
    alert("请输入原始密码");
    document.form1.oldUserPwd.focus();
    return false;
  }else if(document.form1.newUserPwd.value=="")
  {
    <!--提示用户输入新密码-->
    alert("请输入新密码");
    document.form1.newUserPwd.focus();
    return false;
  }else if(document.form1.newUserPwd.value!=document.form1.reigthUserPwd.value)
```

```
    {
      <!--判断新密码和确认密码是否一致-->
      alert("新密码和确认密码不一致!");
      return false;
    }
    else if(document.form1.reigthUserPwd.value=="")
    {
      <!--提示用户输入确认密码-->
      alert("请输入确认密码");
      document.form1.reigthUserPwd.focus();
      return false;
    }else
    {
      return true;
    }
}
</script>
</head>
<body bgcolor="#ffffff">
<!--该页面包含 index.jsp 页面-->
<jsp:include flush="false" page="index.jsp"/>
<center>
  <!--用户提交的信息将提交给 changeAction 类处理-->
    <form name="form1" action="changeAction.do" method="POST">
    <table>
      <tr>
        <td>            用户
          <font color="red">${sessionScope.user.userName}            </font>
          欢迎您登录
        </td>
      </tr>
      </table>
      <br />
      <br />
      <c:if test="${empty requestScope.chengeresult}">
      <table>
      <tr>
      <td>原始密码: 
      <!--输入用户原始密码文本框-->
      <input type="password" name="oldUserPwd" /></td>
      </tr>
          <tr>
      <td>新密码:  
      <!--输入用户新密码文本框-->
      <input type="password" name="newUserPwd" /></td>
      </tr>
          <tr>
      <td>确认密码: 
      <!--提示用户确认密码文本框-->
```

```
        <input type="password" name="reigthUserPwd" /></td>
        </tr>
        <tr>
        <td>
        <!-- “修改” 按钮-->
        <input type="submit" onclick="check()" value="修改" />
        <!-- “重置” 按钮-->
        <input type="reset" value="重置" /></td>
        </tr>
    </table>
    </c:if>
    <c:if test="${requestScope.chengeresult==true}">
    <table>
    <tr>
    <td><font color="red">恭喜密码修改成功!下次可以使用新密码登录!</font></td>
    </tr>
    <tr>
    <td><a href="index.jsp">继续浏览</a></td>
    <td><a href="outAction.do">重新登录</a></td>
    </tr>
    </table>
    </c:if>
        <c:if test="${requestScope.chengeresult==false}">
          <table>
    <tr>
    <td><font color="red">原始密码不正确,请重新输入,修改失败!</font></td>
    </tr>
    <tr>
    <td><a href="chenge.jsp">继续修改</a></td>
    <td><a href="index.jsp">放弃修改</a></td>
    </tr>
    </table>
    </c:if>
    </form>
</center>
</body>
</html>
```

12.7.7　创建编辑 CD 信息页面

编辑 CD 信息页面可以对查询到的某条 CD 信息进行编辑操作，修改后的新信息可以保存到数据库中。

其中的<form action="editAction.do?id=${requestScope.CDDate.cdId}&action=update"method = "POST">语句表明与 edit.jsp 页面相对应的 editAction。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“edit.jsp”文件。

（2）编辑 edit.jsp 文件，输入如下代码：

```
<%@page contentType="text/html; charset=GBK"%>
<!--声明和加载 Struts 标签库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<html>
<head>
<title>编辑 CD 信息</title>
</head>
<!--页面加载时执行查询操作-->
<body bgcolor="#ffffff" onunload="javascript:opener.location.href='selectAction.do';">
<c:if test="${empty requestScope.result}">
  <!--该表单的内容将提交给 editAction 来处理-->
  <form action="editAction.do?id=${requestScope.CDDate.cdId}&action=update"method= "POST">
  <table>
    <tr>
      <td>        编号
         
        <!--显示“编号”文本框-->
        <input    type="text"    name="cdId"    size="2"    value="${requestScope.CDDate.cdId}"
readonly/>
      </td>
    </tr>
    <tr>
      <td>        名字
         
        <!--显示“名字”文本框-->
        <input type="text" name="cdName" value="${requestScope.CDDate.cdName}"/>
      </td>
    </tr>
    <tr>
      <td>        公司
         
        <!--显示“公司”名称文本框-->
        <input type="text" name="cdCompany" value="${requestScope.CDDate.cdCompany}"/>
      </td>
    </tr>
    <tr>
      <td>        歌手
         
         <!--显示“歌手”名称文本框-->
        <input type="text" name="cdAlbum" value="${requestScope.CDDate.cdAlbum}"/>
      </td>
    </tr>
    <tr>
```

```
        <td>          类型
           
          <!--数据库的连接信息-->
          <sql:setDataSource                driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyData"   user="sa"   password=""
var="db" scope="request"/>
          <!--从数据库中进行查询-->
          <sql:query var="query" dataSource="${db}">select * from CDtype</sql:query>
          <select name="cdType">
            <c:forEach var="type" items="${query.rows}">
              <c:if test="${type.display==requestScope.CDDate.cdType}">
                <option value="${type.display}">${type.display}              </option>
              </c:if>
            </c:forEach>
            <c:forEach var="type" items="${query.rows}">
              <c:if test="${type.display!=requestScope.CDDate.cdType}">
                <option value="${type.display}">${type.display}              </option>
              </c:if>
            </c:forEach>
          </select>
        </td>
      </tr>
      <tr>
        <td>
         <!--“更新”按钮-->
          <input type="submit" value="更新"/>
         <!--“取消”按钮-->
          <input type="button" onclick="javascript:window.close(this);" value="取消"/>
        </td>
      </tr>
    </table>
    </form>
</c:if>
<c:if test="${requestScope.result==true}">
    <center>
      <table>
        <tr>
          <td>
            <font color="red">恭喜更新成功</font>
          </td>
        </tr>
        <tr>
          <td>
            <a href="editAction.do?id=${requestScope.CDBean.cdId}">重新操作</a>
          </td>
          <td>
            <a onclick="javascript:opener.location.href='selectAction.do';javascript:window.close
(this);">确认</a>
          </td>
```

```
            </tr>
        </table>
    </center>
</c:if>
<c:if test="${requestScope.result==false}">
    <center>
        <table>
            <tr>
                <td>
                    <font color="red">抱歉更新失败</font>
                </td>
            </tr>
            <tr>
                <td>
                    <a href="editAction.do?id=${requestScope.CDDate.cdId}">重新操作</a>
                </td>
                <td>
                    <a onclick="javascript:window.close(this);">离开</a>
                </td>
            </tr>
        </table>
    </center>
</c:if>
</body>
</html>
```

12.7.8　删除 CD 信息

页面 del.jsp 实现删除 CD 信息的功能，其中的

```
<a href="delAction.do?id=${param.id}">确认</a>
```

语句表明 del.jsp 页面与 delAction 动作相对应。

跟我做

（1）在“cdbox”工程中的“pages”文件夹下创建“del.jsp”文件。
（2）编辑 del.jsp 文件，输入如下代码：

```
<%@ page contentType="text/html; charset=GBK" %>
<!--声明和加载 Struts 标签库-->
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>
删除 CD 信息
</title>
</head>
<body bgcolor="#ffffff">
<c:if test="${empty requestScope.delresult}">
```

```
    <center>
      <table>
        <tr>
          <td>
            <!--提示用户是否删除该条 CD 信息-->
            <font color="red">删除该条 CD 信息？ </font>
          </td>
        </tr>
        <tr>
          <td>
            <!--确认删除-->
            <a href="delAction.do?id=${param.id}">确认</a>
          </td>
          <td>
           <!--取消删除-->
            <a onclick="javascript:window.close(this);">取消</a>
          </td>
        </tr>
      </table>
    </center>
</c:if>
<c:if test="${requestScope.delresult==false}">
    <center>
      <table>
        <tr>
          <td>
            <font color="red">抱歉删除失败</font>
          </td>
        </tr>
        <tr>
          <td>
           <!--关闭“删除”操作窗口-->
            <a onclick="javascript:window.close(this);">离开</a>
          </td>
        </tr>
      </table>
    </center>
</c:if>
<c:if test="${requestScope.delresult==true}">
    <center>
      <table>
        <tr>
          <td>
            <font color="red">恭喜!删除成功</font>
          </td>
        </tr>
        <tr>
          <td>
            <a onclick="javascript:opener.location.href='selectAction.do';javascript:window.close
```

```
(this);"">确认</a>
            </td>
         </tr>
      </table>
   </center>
</c:if>
</body>
</html>
```

12.8　创建 ActionForm

ActionForm 代表了由浏览器所传递过来的数据。每个 ActionForm 中定义的属性应该与页面中表单的控件或者说页面中所提交的数据相对应。这样，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。所有的 ActionForm 类都继承于 org.apache.struts.validator.ValidatorForm 类。

12.8.1　创建添加 CD 信息的 ActionForm

AddCDActionForm 类为典型的 JavaBean，包括歌手、公司名称和 CD 名称 3 个属性，对于每个属性都包括 getter/setter 方法。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。

跟我做

（1）右击“cdbox”工程，在快捷菜单中选择【新建】|【源文件夹】命令，打开【新建源文件夹】对话框，在【文件夹名】文本框中输入“src”，单击【确定】按钮，创建名字为“src”的源文件夹。

（2）右击“src”源文件夹，在快捷菜单中选择【新建】|【类】命令，打开【新建 Java 类】对话框。在【名称】文本框中输入“AddCDActionForm”。单击【超类】文本框后面的【浏览】按钮，打开【超类选择】对话框。

（3）选择 org.apache.struts.validator.ValidatorForm 类，如图 12-19 所示，单击【确定】按钮。单击【新建 Java 类】中的【完成】按钮，创建名字为 AddCDActionForm.java 的类。



图 12-19　超类选择

（4）编辑新创建的 AddCDActionForm 类，输入如下代码：

```
public class AddCDActionForm extends ActionForm {
	//“歌手”属性
	private String cdAlbum;
   //“出版公司”属性
	private String cdCompany;
   //“CD 名称”属性
	private String cdName;

	/**
	 * 取得“歌手”属性值
	 * @return
	 */
	public String getCdAlbum() {
		return cdAlbum;
	}

	/**
	 * 设置“歌手”属性值
	 * @param cdAlbum
	 */
	public void setCdAlbum(String cdAlbum) {
		this.cdAlbum = cdAlbum;
	}

	/**
	 * 设置 CD 名称属性
	 * @param cdName
	 */
	public void setCdName(String cdName) {
		this.cdName = cdName;
	}

	/**
	 * 设置 CD 出版公司名称
	 * @param cdCompany
	 */
	public void setCdCompany(String cdCompany) {
		this.cdCompany = cdCompany;
	}

	/**
	 * 取得 CD 出版公司名称属性
	 * @return
	 */
	public String getCdCompany() {
		return cdCompany;
	}
```

```
        /**
         * 取得 CD 名字属性
         * @return
         */
        public String getCdName() {
                return cdName;
        }

        public ActionErrors validate(ActionMapping actionMapping,
                        HttpServletRequest httpServletRequest) {
                /** @todo: finish this method, this is just the skeleton. */
                return null;
        }

        public void reset(ActionMapping actionMapping,
                        HttpServletRequest servletRequest) {
        }
}
```

12.8.2　创建修改密码的 ActionForm

该类是典型的 JavaBean，包括用户的新密码、用户的旧密码和用户的确认密码 3 个属性，对于每个属性都有其对应的 getter/setter 方法。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。ChangePass ActionForm 用于修改密码。

跟我做

在“src”文件夹下创建 ChangePassActionForm.java 类，编辑该文件，输入如下代码：

```
public class ChangePassActionForm extends ActionForm {
        // 新用户密码属性
        private String newUserPwd;

        // 旧的用户密码属性
        private String oldUserPwd;

        // 用户的确认密码属性
        private String reigthUserPwd;

        /**
         * 取得用户的新密码属性的值
         *
         * @return
         */
        public String getNewUserPwd() {
                return newUserPwd;
        }
```

```
/**
 * 设置用户的新密码属性的值
 *
 * @param newUserPwd
 */
public void setNewUserPwd(String newUserPwd) {
	this.newUserPwd = newUserPwd;
}

/**
 * 设置用户的确认密码的值
 *
 * @param reigthUserPwd
 */
public void setReigthUserPwd(String reigthUserPwd) {
	this.reigthUserPwd = reigthUserPwd;
}

/**
 * 设置用户的旧密码属性的值
 *
 * @param oldUserPwd
 */
public void setOldUserPwd(String oldUserPwd) {
	this.oldUserPwd = oldUserPwd;
}

/**
 * 取得旧的用户密码
 *
 * @return
 */
public String getOldUserPwd() {
	return oldUserPwd;
}

/**
 * 取得确认密码的值
 *
 * @return
 */
public String getReigthUserPwd() {
	return reigthUserPwd;
}

public ActionErrors validate(ActionMapping actionMapping,
		HttpServletRequest httpServletRequest) {
	/** @todo: finish this method, this is just the skeleton. */
```

```
            return null;
        }

        public void reset(ActionMapping actionMapping,
                    HttpServletRequest servletRequest) {
        }
}
```

12.8.3　创建用户登录 ActionForm

该类是典型的 JavaBean 类，包括用户名和密码两个属性，每个属性都有其对应的 getter/setter 方法。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。该 ActionForm 类用于收集用户的系统登录信息。

跟我做

在“src”文件夹下创建 LoginActionForm.java 类，编辑该文件，输入如下代码：

```
public class LoginActionForm extends ActionForm {
        // “用户名”属性
        private String userName;

        // “用户密码”属性
        private String userPwd;

        /**
         * 取得 UserName 属性的值
         *
         * @return
         */
        public String getUserName() {
            return userName;
        }

        /**
         * 设置 UserName 的属性
         *
         * @param userName
         */
        public void setUserName(String userName) {
            this.userName = userName;
        }

        /**
         * 设置用户密码属性
         *
         * @param userPwd
```

```
	 */
	public void setUserPwd(String userPwd) {
		this.userPwd = userPwd;
	}

	/**
	 * 取得用户密码属性的值
	 *
	 * @return
	 */
	public String getUserPwd() {
		return userPwd;
	}

	public ActionErrors validate(ActionMapping actionMapping,
			HttpServletRequest httpServletRequest) {
		/** @todo: finish this method, this is just the skeleton. */
		return null;
	}

	public void reset(ActionMapping actionMapping,
			HttpServletRequest servletRequest) {
	}
}
```

12.8.4　创建用户注册 ActionForm

该类是典型的 JavaBean 类，包括用户名、用户密码和用户确认密码 3 个属性，每个属性都有其对应的 getter/setter 方法，用于收集用户的注册信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。RegActionForm 类实现用户注册。

跟我做

在“src”文件夹下创建 RegActionForm.java 类，编辑该文件，输入如下代码信息：

```
public class RegActionForm extends ActionForm {
	// “确认用户密码”属性
	private String reigthUserPwd;

	// “用户名”属性
	private String userName;

	// “用户密码”属性
	private String userPwd;

	/**
	 * 取得“确认用户密码”属性的值
```

```
 * @return
 */
public String getReigthUserPwd() {
        return reigthUserPwd;
}

/**
 * 设置“确认用户密码”属性的值
 * @param reigthUserPwd
 */
public void setReigthUserPwd(String reigthUserPwd) {
        this.reigthUserPwd = reigthUserPwd;
}

/**
 * 设置“用户密码”属性的值
 * @param userPwd
 */
public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
}

/**
 * 设置“用户名”属性的值
 * @param userName
 */
public void setUserName(String userName) {
        this.userName = userName;
}

/**
 * 取得“用户名”属性的值
 * @return
 */
public String getUserName() {
        return userName;
}

/**
 * 取得用户密码属性的值
 * @return
 */
public String getUserPwd() {
        return userPwd;
}

public ActionErrors validate(ActionMapping actionMapping,
                HttpServletRequest httpServletRequest) {
        /** @todo: finish this method, this is just the skeleton. */
```

```
            return null;
        }

        public void reset(ActionMapping actionMapping,
                    HttpServletRequest servletRequest) {
        }
}
```

12.8.5　创建搜索 CD 信息的 ActionForm

该类是典型的 JavaBean 类，包括页码和搜索关键字属性，对于每个属性都包括 getter/setter 方法，用于搜集用户的输入信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。SearchActionForm 处理用户输入的搜索关键字。

跟我做

在“src”文件夹中创建 SearchActionForm.java 类，编辑该文件，输入如下代码：

```
public class SearchActionForm extends ActionForm {
        // 页码属性
        private String pageText;

        // 搜索关键字属性
        private String selectValue;

        /**
         * 取得页码属性的值
         *
         * @return
         */
        public String getPageText() {
            return pageText;
        }

        /**
         * 设置页码属性的值
         *
         * @param pageText
         */
        public void setPageText(String pageText) {
            this.pageText = pageText;
        }

        /**
         * 设置搜索关键字属性的值
         *
         * @param selectValue
```

```
     */
    public void setSelectValue(String selectValue) {
        this.selectValue = selectValue;
    }

    /**
     * 取得搜索关键字的值
     *
     * @return
     */
    public String getSelectValue() {
        return selectValue;
    }

    public ActionErrors validate(ActionMapping actionMapping,
            HttpServletRequest httpServletRequest) {
        /** @todo: finish this method, this is just the skeleton. */
        return null;
    }

    public void reset(ActionMapping actionMapping,
            HttpServletRequest servletRequest) {
    }
}
```

12.9　使用 Action 类实现控制层

Action 的作用是接受用户的请求，通过调用业务方法实现业务处理的功能。在编写 Action 时必须要继承自 org.apache.struts.action.Action 类，并覆盖父类中的 execute()方法。这里的 execute()方法就是响应用户请求的处理方法，它是被 Struts 框架自动调用的。当用户在页面中提交表单后，Struts 框架就会把用户请求转发给 Action 组件。本节实现 CD 管理系统中的所有 Action 类，实现其控制层。

12.9.1　创建添加 CD 信息 Action

该类继承于 Action 类，其从 AddCDActionForm 中取得 CD 信息，将这些信息实例化一个 CD 类，然后通过 DBManager 的 addCD()方法将 CD 信息提交到数据库中。

跟我做

（1）在“cdbox”工程中创建 AddCDAction 类，继承于 org.apache.struts.action.Action 类。

（2）编辑 AddCDAction.java 类，输入如下代码：

```
public class AddCDAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
```

```
                HttpServletRequest request, HttpServletResponse response) {
            // 将参数 form 进行类型转换
            AddCDActionForm addCDForm = (AddCDActionForm) form;
            //声明操作数据库的类
            DBManager dbManager = new DBManager();
            //从 form 中取得 CD 名称属性
            String cdName = addCDForm.getCdName();
            //从 form 中取得 CD 公司名称属性
            String cdCompany = addCDForm.getCdCompany();
            //从 form 中取得歌手名称属性
            String cdAlbum = addCDForm.getCdAlbum();
            //从 form 中取得 CD 类型属性
            String cdType = request.getParameter("cdType");
            //实例化一个 CD 类的实例
            CD cd = new CD();
            //设置 cd 的“歌手”属性
            cd.setCdAlbum(cdAlbum);
            //设置 cd 的“公司名称”属性
            cd.setCdCompany(cdCompany);
            //设置 cd 的“cd 名称”属性
            cd.setCdName(cdName);
            //设置 cd 的“cd 类型”属性
            cd.setCdType(cdType);
            //通过 dbManager 将新的 CD 信息插入到数据库中
            boolean result = dbManager.addCD(cd);
            //为 request 对象设置“addresult”属性
            request.setAttribute("addresult", result);
            //关闭 dbManager
            dbManager.close();
            return mapping.findForward("add");
      }
}
```

12.9.2　创建修改用户密码 Action

该类首先通过原始密码判断用户的合法性，然后将新的密码保存到数据库中，并将保存结果放入 request 中。其中的 DBManager 封装了所有与数据库相关的操作，通过

```
result = bm.updateUserPwd(user);
```

语句实现了更新用户的密码信息。

跟我做

在“cdbox”工程中创建 ChangePassAction 类，完成修改用户密码的逻辑。编辑该文件，输入如下代码：

```
/**
 * 处理用户修改密码的操作
```

```
 *
 */
public class ChangePassAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 将参数 form 进行类型转换
        ChangePassActionForm changePassForm = (ChangePassActionForm) form;
        // 声明 DBManager 对象
        DBManager bm = new DBManager();
        boolean result;
        //取得 request 中的 User 对象
        User user = (User) request.getSession().getAttribute("user");
        //判断用户输入的原始密码是否合法
        if (user.getUserPwd().equals(changePassForm.getOldUserPwd())) {
            //为 user 设置 userpwd 属性
            user.setUserPwd(changePassForm.getNewUserPwd());
            //通过 bm 更新用户的密码信息
            result = bm.updateUserPwd(user);
        } else {
            result = false;
        }
        //将更新结果保存到 request 中
        request.setAttribute("chengeresult", result);
        bm.close();
        return mapping.findForward("chenge");
    }
}
```

12.9.3　创建删除 CD 信息 Action

该类覆盖了 Action 类的 execute()方法，根据 CD 的 id 通过如下语句

```
boolean result = bm.deleteCD(Long.parseLong(request
                            .getParameter("id")));
```

从数据库中删除 CD 信息，并将删除的结果保存到 request 中。

跟我做

在“cdbox”工程中创建 DelCDAction 类，来完成删除 CD 信息的功能。编辑该文件，输入如下代码信息：

```
public class DelCDAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
          //从 request 对象中取得 id 属性的值
        if (request.getParameter("id") != null) {
            //创建 DBManager 对象
            DBManager bm = new DBManager();
           //调用 bm 中的 deleteCD 从数据库中删除特定的 CD 信息
```

```
                boolean result = bm.deleteCD(Long.parseLong(request
                        .getParameter("id")));
                //将查询结果存入到 delresult 中
                request.setAttribute("delresult", result);
                bm.close();
            }

            return mapping.findForward("del");
        }
}
```

12.9.4　创建编辑 CD 信息 Action

该类从 request 中取得 id 的值，显示该条 CD 信息的值。用户可以修改这些值，然后将更新后的信息通过如下语句

```
boolean result = bm.updataCD(cd);
```

保存到数据库中。

跟我做

在“cdbox”工程中创建 EditCDAction 类，来完成编辑 CD 信息的功能。编辑该文件，输入如下代码信息：

```
public class EditCDAction extends Action {
        public ActionForward execute(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response) {
            // 创建 DBManager 对象
            DBManager bm = new DBManager();
            if (request.getParameter("id") != null) {
                // 从 request 对象中取得 id 属性的值
                long id = Long.parseLong(request.getParameter("id"));
                // CD 对象
                CD cd = null;
                // 判断 request 中的 action 对象是否为 null
                if (request.getParameter("action") != null) {
                    //判断 request 中的 action 是否为 updata
                    if (request.getParameter("action").equals("updata")) {
                        //创建 CD 对象
                        cd = new CD();
                        //设置 cd 的歌手属性
                        cd.setCdAlbum(request.getParameter("cdAlbum"));
                        //设置 cd 的公司名称属性
                        cd.setCdCompany(request.getParameter("cdCompany"));
                        //设置 CD 名称属性
                        cd.setCdName(request.getParameter("cdName"));
                        //设置账号 CD 类型属性
                        cd.setCdType(request.getParameter("cdType"));
```

```
                    //设置 CDid 属性
                    cd.setCdId(id);
                    //通过 bm 更新 CD 的信息
                    boolean result = bm.updataCD(cd);
                    //将结果保存到 result 中
                    request.setAttribute("result", result);

                }
            } else {
                //如果操作不为 updata 就会从数据库中查询 CD 信息
                cd = bm.getCD(id);
            }
            //在 request 中设置 CDDate 属性
            request.setAttribute("CDDate", cd);
        }
        return mapping.findForward("edit");
    }
}
```

12.9.5　创建用户登录 Action

该类覆盖了 Action 类的 execute()方法，从 ActionForm 中取得用户名和用户密码信息，然后通过 DBManager 的 checkUser()方法判断用户的合法性。从而对于合法用户实现系统的登录。

跟我做

在“cdbox”工程中创建 LoginAction 类，用来处理用户的登录信息。编辑该文件，输入如下代码信息：

```
public class LoginAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 对 form 参数进行类型转换
        LoginActionForm loginForm = (LoginActionForm) form;
        String forword = "login";
        //创建 User 对象
        User user = new User();
        //设置用户名属性
        user.setUserName(loginForm.getUserName());
        //设置用户密码属性
        user.setUserPwd(loginForm.getUserPwd());
        //创建 DBManager 对象
        DBManager bm = new DBManager();
        //检查该用户是否已经存在
        if (bm.checkUser(user)) {
            //在 request 中保存 user 属性
            request.getSession().setAttribute("user", user);
```

```
                forword = "index";
            }
            //关闭 bm
            bm.close();
            return mapping.findForward(forword);
        }
}
```

12.9.6　创建用户注销 Action

该类覆盖了 Action 类的 execute()方法，从 Session 中删除 user、selectList 等属性。OutAction 实现用户注销功能。

跟我做

在“cdbox”工程中创建 OutAction 类，实现用户注销功能。编辑 OutAction.java，输入如下代码信息：

```
public class OutAction extends Action {
        public ActionForward execute(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response) {
            //从 session 中删除“user”属性
            request.getSession().removeAttribute("user");
            //从 session 中删除“selectList”属性
            request.getSession().removeAttribute("selectList");
            //将 request 中的 user 属性置 null
            request.getSession().setAttribute("user", null);
            return mapping.findForward("login");
        }
}
```

12.9.7　创建用户注册 Action

该类实现新用户的注册功能，覆盖了 Action 类的 execute()方法，从 ActionForm 中取得用户信息，然后通过如下语句

```
boolean result = bm.addUser(user);
```

将新的用户信息保存到数据库中。

跟我做

在“cdbox”工程中创建 RegAction 类，实现新用户的注册功能。编辑 RegAction.java 类，输入如下代码信息：

```
public class RegAction extends Action {
        public ActionForward execute(ActionMapping mapping, ActionForm form,
                HttpServletRequest request, HttpServletResponse response) {
```

```
        //对 form 进行类型转换
        RegActionForm regForm = (RegActionForm) form;
        //创建 DBManager 对象
        DBManager bm = new DBManager();
        //取得用户名属性的值
        String userName = regForm.getUserName();
        //取得用户名密码属性
        String userPwd = regForm.getUserPwd();
        //创建 User 对象
        User user = new User();
        //设置 User 对象的 userName 属性
        user.setUserName(userName);
        //设置 user 对象的 userPwd 属性
        user.setUserPwd(userPwd);
        //通过 bm 将新增加的用户信息添加到数据库中
        boolean result = bm.addUser(user);
        //在 request 中设置 regresult 值
        request.setAttribute("regresult", result);
        //关闭 bm
        bm.close();
        return mapping.findForward("reg");

    }
}
```

12.9.8　创建 CD 搜索 Action

该类覆盖了 Action 类的 execute()方法，根据查询条件从数据库中查询满足条件的 CD 信息，并且将查询结果以分页的形式来显示，用户可以指定要查看的页码。SearchAction 类实现搜索 CD 信息的功能。

跟我做

在“cdbox”工程中创建 SearchAction 类，实现搜索 CD 信息的功能。编辑 SearchAction.java 类，输入如下代码信息：

```
public class SearchAction extends Action {
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 对 form 参数进行类型转换
        SearchActionForm selectForm = (SearchActionForm) form;
        // 创建 DBManager 对象
        DBManager bm = new DBManager();
        int page = 1;
        // 判断“搜索关键字”是否为空
        if (selectForm.getSelectValue() != null) {
            if (request.getParameter("page") == null) {
                page = 1;
```

```
                } else {

                    // 判断要跳转的页码
                    page = Integer.parseInt(request.getParameter("page"));
                }

                if (selectForm.getPageText() != null) {
                    // 从 form 中读取用户要跳转的页码
                    page = Integer.parseInt(selectForm.getPageText());
                }
                if (request.getParameter("action") != null) {
                    // 如果选择了“首页”，则 page=1
                    if (request.getParameter("action").equals("frist")) {
                        page = 1;
                    } else if (request.getParameter("action").equals("last")) {
                        page = bm.PAGECOUNT;
                    } else if (request.getParameter("action").equals("back")) {
                        // 如果选择了“后退”，将 page 减 1
                        page -= 1;
                    } else if (request.getParameter("action").equals("frist")) {
                        // 如果选择了“前进”，将 page 加 1
                        page += 1;
                    }
                }
                // 通过 bm 从数据库中查询符合条件的 CD 信息
                List list = bm.searchCD(selectForm.getSelectValue(), page, 10);
                // 将查询结果保存在 session 中
                request.getSession().setAttribute("selectList", list);
                int pagecount = bm.PAGECOUNT;
                // 将当前页码保存到 request 中
                request.setAttribute("pagecount", pagecount);
                // 将 selectList 属性从 session 中删除
                request.getSession().removeAttribute("selectList");
                // 将 selectValue 保存到 request 中
                request.setAttribute("selectValue", selectForm.getSelectValue());
                // 将 page 保存到 request 中
                request.setAttribute("page", page);
                // 将 pagecount 保存到 request 中
                request.setAttribute("pagecount", pagecount);

            } else {
                request.getSession().removeAttribute("selectList");
            }
            // 关闭 bm
            bm.close();
            return mapping.findForward("select");
        }
}
```

12.10　生成 Struts 配置文件

Struts 的核心是控制器，即 ActionServlet，而 ActionServlet 的核心就是 Struts-config.xml，Struts-config.xml 集中了所有页面的导航定义。掌握 Struts-config.xml 是掌握 Struts 的关键所在。本节创建 Struts 的配置文件，将 JSP 页面、Action 等组件结合起来。

跟我做

在“cdbox”工程中创建“struts-config.xml”文件，在文件中输入如下代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
  <!--FromBean 的定义-->
  <form-beans>
    <!--login.jsp 对应的 FormBean-->
    <form-bean
        name="loginActionForm"
        type="webapp.LoginActionForm" />
     <!--select.jsp 对应的 FormBean-->
    <form-bean
        name="selectActionForm"
        type="SelectActionForm" />
     <!--change.jsp 对应的 FormBean-->
    <form-bean
        name="changeActionForm"
        type="ChengeActionForm" />
        <!--add.jsp 对应的 FormBean-->
    <form-bean
        name="addActionForm"
        type="AddActionForm" />
        <!--reg.jsp 对应的 FormBean-->
    <form-bean
        name="regActionForm"
        type="RegActionForm" />
  </form-beans>
    <!--定义全局页面的跳转-->
  <global-forwards>
    <!--定义名字为 login 的全局跳转-->
    <forward
        name="login"
        path="/login.jsp" />
    <!--定义名字为 index 的全局跳转-->
    <forward
```

```
            name="index"
            path="/index.jsp" />
        <!--定义名字为 select 的全局跳转-->
        <forward
            name="select"
            path="/select.jsp" />
        <!--定义名字为 edit 的全局跳转-->
        <forward
            name="edit"
            path="/edit.jsp" />
        <!--定义名字为 del 的全局跳转-->
        <forward
            name="del"
            path="/del.jsp" />
        <!--定义名字为 change 的全局跳转-->
        <forward
            name="change"
            path="/change.jsp" />
        <!--定义名字为 add 的全局跳转-->
        <forward
            name="add"
            path="/add.jsp" />
        <!--定义名字为 reg 的全局跳转-->
        <forward
            name="reg"
            path="/reg.jsp" />
    </global-forwards>
<!-- Action 映射定义  -->
    <action-mappings>
      <!--  定义 path 为"/ loginAction"的 Action -->
        <action
             input="/login.jsp"
             name="loginActionForm"
             path="/loginAction"
             scope="request"
             type="webapp.LoginAction"
             validate="true" />
             <!--  定义 path 为"/ selectAction"的 Action -->
        <action
             input="/select.jsp"
             name="selectActionForm"
             path="/selectAction"
             scope="request"
             type="SelectAction"
             validate="true" />
             <!--  定义 path 为"/ editAction"的 Action -->
        <action
             path="/editAction"
             type="EditAction" />
```

```
        <!-- 定义 path 为"/ delAction"的 Action -->
    <action
        path="/delAction"
        type="DelAction" />
        <!-- 定义 path 为"/ changeAction"的 Action -->
    <action
        input="/change.jsp"
        name="changeActionForm"
        path="/changeAction"
        scope="request"
        type="ChangeAction"
        validate="true" />
        <!-- 定义 path 为"/ outAction"的 Action -->
    <action
        path="/outAction"
        type="OutAction" />
        <!-- 定义 path 为"/addAction"的 Action -->
    <action
        input="/add.jsp"
        name="addActionForm"
        path="/addAction"
        scope="request"
        type="AddAction"
        validate="true" />
        <!-- 定义 path 为"/ regAction"的 Action -->
    <action
        input="/reg.jsp"
        name="regActionForm"
        path="/regAction"
        scope="request"
        type="RegAction"
        validate="true" />
  </action-mappings>
  <message-resources parameter="ApplicationResources" />
</struts-config>
```

12.11　系统的 Tomcat 部署

本节将介绍一种快速部署 Tomcat 的方法。

12.11.1　CDManagerFilter 的创建

跟我做

（1）在“cdbox”工程中创建“CDManagerFilter.java”文件，编辑该文件，输入如下代码信息：

```
public class CDManagerFilter extends HttpServlet implements Filter {
	private FilterConfig filterConfig;

	// 处理传入的 FilterConfig 对象
	public void init(FilterConfig filterConfig) throws ServletException {
		this.filterConfig = filterConfig;
	}

	// 处理 request/response
	public void doFilter(ServletRequest request, ServletResponse response,
			FilterChain filterChain) {
		HttpServletRequest httpRequest = (HttpServletRequest) request;
		HttpServletResponse httpResponse = (HttpServletResponse) response;
		// 取得 requestURI
		String url = httpRequest.getRequestURI();
		// 取得 user 对象
		User user = (User) (httpRequest.getSession().getAttribute("user"));
		try {
			request.setCharacterEncoding("GBK");
			if (user == null) {

				if (url.indexOf("login.jsp") != -1
						|| url.indexOf("loginAction.do") != -1
						|| url.indexOf("reg.jsp") != -1
						|| url.indexOf("regAction.do") != -1) {
					// 跳转到下一个 filter
					filterChain.doFilter(httpRequest, response);

				} else {
					// 重定向到 login.jsp 页面中
					httpResponse.sendRedirect("/cdbox/login.jsp");
				}
			} else {

				filterChain.doFilter(request, response);
			}
		} catch (ServletException ex) {
			ex.printStackTrace();
		} catch (IOException ex) {
			ex.printStackTrace();
		}
	}

	// Clean up resources
	public void destroy() {
	}
}
```

该 Filter 截获 request 中的 URL，并对其进行分析。

（2）右击“cdbox”工程，在快捷菜单中选择【新建】|【文件】命令，创建“web.xml”文件。编辑该文件，输入如下代码信息：

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/
xml/ns/j2ee/web-app_2_4.xsd"
  version="2.4"><display-name>cdbox</display-name>
  <jsp-config>
    <taglib>
      <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
    </taglib>
    <taglib>
      <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
    <taglib>
      <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>
  </jsp-config>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <init-param>
      <param-name>debug</param-name>
      <param-value>2</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <filter>
    <filter-name>myfilter</filter-name>
    <filter-class>MyFilter</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>myfilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
</web-app>
```

12.11.2　Tomcat 部署

跟我做

（1）回顾第 7 章下载的 Struts，将 C:\jakarta-struts-1.2.4\webapps\struts-blank.war 文件复制到 %Tomcat%/webapps 目录下，启动 Tomcat，该 war 文件将自动解压，在 webapps 目录下生成“struts-blank”文件夹。

（2）将“struts-blank”文件夹重新命名为“cdbox”。

（3）将 Eclipse 中“cdbox”工程下“pages”文件夹中的所有 jsp 页面复制到“cdbox”文件夹下。

（4）复制“cdbox”工程中的“struts-config.xml”文件覆盖掉 cdbox\WEB-INF 目录下的 struts-config.xml 文件。

（5）将 Eclipse 切换至资源透视图，将“cdbox”工程中 bin 目录下的所有 class 文件复制到 OnlineBBS\WEB-INF\classes 目录下。

（6）复制“cdbox”工程的 MessageResource.properties 文件到%Tomcat%\webapps\cdbox\WEB-INF\classes 目录下，覆盖掉原先的同名文件。

（7）复制“cdbox”工程下的“web.xml”文件到 %Tomcat%\webapps\cdbox\WEB-INF\目录下，覆盖掉原先的同名文件。

经过上面的步骤，完成了整个工程的部署，重启 Tomcat 即可。

第 13 章　综合实例——网上书店管理应用系统

网上书店管理应用系统是一个典型的电子商务应用实例。它综合了 JSP、JSTL、JDBC 和 Struts 等技术。从技术层面来说，该系统既涵盖了前端的应用客户和 Web 客户，又详细描述了 Web 组件等中间件技术，也包含了 Web 服务器、数据库、生成和部署工具等。从业务层面来说，该系统包括用户登录、图书列表、图书管理、订单管理等功能。

本章通过该综合实例的介绍，使读者熟练掌握在 Eclipse 环境下开发典型的 J2EE 应用实例的具体过程。

13.1　需求分析

网上书店系统是一个复杂的应用系统，分为后台管理和前台展示两部分。后台管理包括图书的添加、图书的分类、用户的管理等功能。前台部分是一个界面友好的网上书店，顾客在这个书店中可以选择自己喜欢的图书并放进购物车中，完成购买图书的功能。

本节对网上书店系统进行需求分析，明确整个系统所包含的模块、每个模块所实现的业务功能、系统的数据库设计和系统需要处理的业务流程。

13.1.1　后台管理系统

后台管理系统是网上书店的管理模块。下面介绍其应该实现的功能。

（1）图书管理：包括新书的添加、书籍类别的添加和查看所有图书的功能。

（2）订单查看：可以查看所有订单的详细情况并对每个订单进行处理。

（3）用户管理：对系统的所有用户进行管理，包括用户注册信息的修改和用户的增删。

（4）Admin 管理：可以为系统添加管理员以及修改管理员密码。

13.1.2　前台展示系统

前台展示系统提供界面友好的图书分类展示，提供顾客完成订购图书的功能。其应该实现如下功能：

（1）图书的分类管理：在后台管理系统中管理员可以将所有的图书分类，前台展示系统按照这个分类将所有图书展现给用户，方便用户查找书籍。

（2）特价商品的展示。

（3）销售排行：对所有的图书进行销售情况排行。

（4）用户登录和注册功能：网上书店系统的用户管理是至关重要的。只有注册用户才有购买图书的权限。

（5）图书搜索功能：根据关键字对数据库中的所有图书进行搜索。

13.1.3　数据的存储

网上书店管理应用系统的数据库设计是至关重要的，需要保存的系统信息包括图书信息、用户信息、类型信息和分类信息等。

本小节设计网上书店系统的数据库结构，所有表格如表 13-1 所示。

表 13-1　系统的所有表格

表　编　号	表　　名
TBL001	admin
TBL002	BookStep
TBL003	city
TBL004	information
TBL005	OrderForm
TBL006	Step
TBL007	Type
TBL008	users

各表具体含义详细描述如下所示。

admin 表存储系统管理员的登录信息，userId 为管理员的 ID 编号，userName 为管理员的用户名，userPwd 为管理员的密码。其具体表结构如表 13-2 所示。

表 13-2　admin表

名　　称	关　键　字	类　　型	长　　度	值　　域	默　认　值	备　　注
userId	P	自动增长	8	Long		
userName		VC	50			
userPwd		VC	50			

BookStep 表存储用户级别信息，其中 display 为用户级别值，BookStepId 为该用户级别设置的 ID 编号。其具体表结构如表 13-3 所示。

表 13-3　BookStep表

名　　称	关　键　字	类　　型	长　　度	值　　域	默　认　值	备　　注
BookStepId	P	int	4			
display		VC	50			

city 表存储用户的所在城市信息，其中 display 属性显示城市的名称，CityId 为不同城市设置的 ID 编号。其具体表结构如表 13-4 所示。

表 13-4　city表

名　　称	关 键 字	类　　型	长　　度	值　　域	默 认 值	备　　注
CityId	P	int	4			
display		VC	50			

information 表是保存书籍的主要信息，包括书籍名称、出版社、出版日期等信息，其中的 BookId 是为书籍设定的 ID 编号。其具体表结构如表 13-5 所示。

表 13-5　information表

名　　称	关 键 字	类　　型	长　　度	值　　域	默 认 值	备　　注
BookId	P	bigint	8			
BookName		VC	50			
BookPenster		VC	50			
BookCompany		bigint	8			
BookSynopsis		VC	50			
BookStorage		VC	50			
BookSell		VC	50			
BookDate		bigint	8			
BookPrice		VC	50			
BookType		VC	50			
bookImage		VC	50			
BookStep		VC	50			

OrderForm 表是保存用户订单的主要信息，包括用户 ID、书籍 ID、书籍单价等重要信息，其中的 OrderId 是为订单设定的 ID 编号。其具体表结构如表 13-6 所示。

表 13-6　OrderForm表

名　　称	关 键 字	类　　型	长　　度	值　　域	默 认 值	备　　注
OrderId	P	bigint	8			
UserId		VC	50			
BookId		VC	50			
amount	P	bigint	8			
OrderData		VC	50			
money		VC	50			

Step 表保存用户等级信息，其中的 StepId 是为等级设定的 ID 编号。其具体表结构如表 13-7 所示。

表 13-7　Step表

名　称	关　键　字	类　型	长　度	值　域	默　认　值	备　注
StepId	P	int	4			
display		VC	50			

Type 表保存用户类型信息，其中的 typeId 是为类型设定的 ID 编号。其具体表结构如表 13-8 所示。

表 13-8　type表

名　称	关　键　字	类　型	长　度	值　域	默　认　值	备　注
typeId	P	int	4			
display		VC	50			

users 表保存着所有的用户信息，包括用户名、用户密码、用户真实姓名等信息，其中的 userId 是为用户设定的 ID 编号。其具体表结构如表 13-9 所示。

表 13-9　users表

名　称	关　键　字	类　型	长　度	值　域	默　认　值	备　注
userId	P	bigint	8			
userName		VC	50			
userPwd		VC	50			
realName	P	bigint	8			
UserSex		VC	50			
UserPhone		VC	50			
Usermail		VC	50			
UserCity	P	bigint	8			
UserAdds		VC	50			
UserCode		VC	50			
Usercard		VC	50			
UserWork		VC	50			
UserStep		VC	50			
UserAge		VC	50			

13.1.4　系统所有处理的描述

网上书店系统主要包括对用户管理的操作和对书籍管理的操作，本小节详细分析系统的业务需求，列出了系统中需要具备的所有操作，如表 13-10 所示。

表 13-10 系统的所有处理

处 理 编 号	处 理 名
1	用户登录
2	用户注销
3	修改密码
4	添加图书信息
5	添加图书分类
6	查询图书信息
7	删除图书信息

系统应该具备用户登录、用户注销等具体操作，各处理详细描述如下所示。

用户登录时提供用户名和用户密码，系统验证用户的合法性，并且将验证结果返回给用户。其描述如表 13-11 所示。

表 13-11 用户登录

处 理 名	用 户 登 录
处理编号	1
输入数据流	用户名+用户密码
输出数据流	提示用户登录成功或失败
处理逻辑	根据用户登录信息查询用户信息表，从而验证用户的合法性

用户注销操作是用户从系统中注销，重新返回用户登录页面。其描述如表 13-12 所示。

表 13-12 用户注销

处 理 名	用户注销
处理编号	2
输入数据流	无
输出数据流	用户从系统中注销，重新返回用户登录页面
处理逻辑	用户从系统中注销，重新返回用户登录页面

修改密码操作是用户提供用户名、用户密码、新密码和确认密码等信息，系统验证用户提供的新密码和确认密码是否一致，如果一致就将新密码保存到数据库中。其详细描述如表 13-13 所示。

表 13-13 修改密码

处 理 名	修 改 密 码
处理编号	3
输入数据流	用户名+用户密码+新密码+确认新密码
输出数据流	新密码更新到用户信息表中
处理逻辑	验证用户提供的新密码和确认新密码是否一致，如果一致就将新密码保存到数据库中

添加图书信息操作将用户通过表单提交的图书名称、出版社、作者等信息保存到数据库中。其描述如表 13-14 所示。

表 13-14　添加图书信息

处理名	添加图书信息
处理编号	4
输入数据流	图书名称、出版社、作者等信息
输出数据流	将新的图书信息保存到数据库中
处理逻辑	将用户通过表单提交的图书信息保存到数据库中

添加图书分类信息操作将用户通过表单提交的图书分类信息保存到数据库中。其描述如表 13-15 所示。

表 13-15　添加图书分类信息

处　理　名	添加图书分类
处理编号	5
输入数据流	分类名称
输出数据流	将新的图书分类信息保存到数据库中
处理逻辑	将用户通过表单提交的图书分类信息保存到数据库中

查询图书信息操作根据用户提供的查询关键字从数据库中查询符合条件的图书信息。其描述如表 13-16 所示。

表 13-16　查询图书信息

处　理　名	查询图书信息
处理编号	6
输入数据流	查询关键字
输出数据流	数据库中满足条件的图书信息
处理逻辑	根据用户输入的查询关键字从数据库中查询符合条件的图书信息

删除图书信息操作根据输入的图书名称和作者名称信息从数据库中删除对应的图书信息。其描述如表 13-17 所示。

表 13-17　删除图书信息

处　理　名	删除图书信息
处理编号	7
输入数据流	图书名称、作者名称
输出数据流	无
处理逻辑	将该条图书信息从数据库中删除

13.2　系统的运行效果

网上书店系统分为后台管理和前台展示两大部分。本节介绍整个系统的运行效果，从而从整体上对系统的功能有所了解。

跟我做

（1）启动 Tomcat 服务器。

（2）打开浏览器，在地址栏中输入 http://localhost:8080/shop/admin/index1.jsp，进入如图 13-1 所示的登录页面。

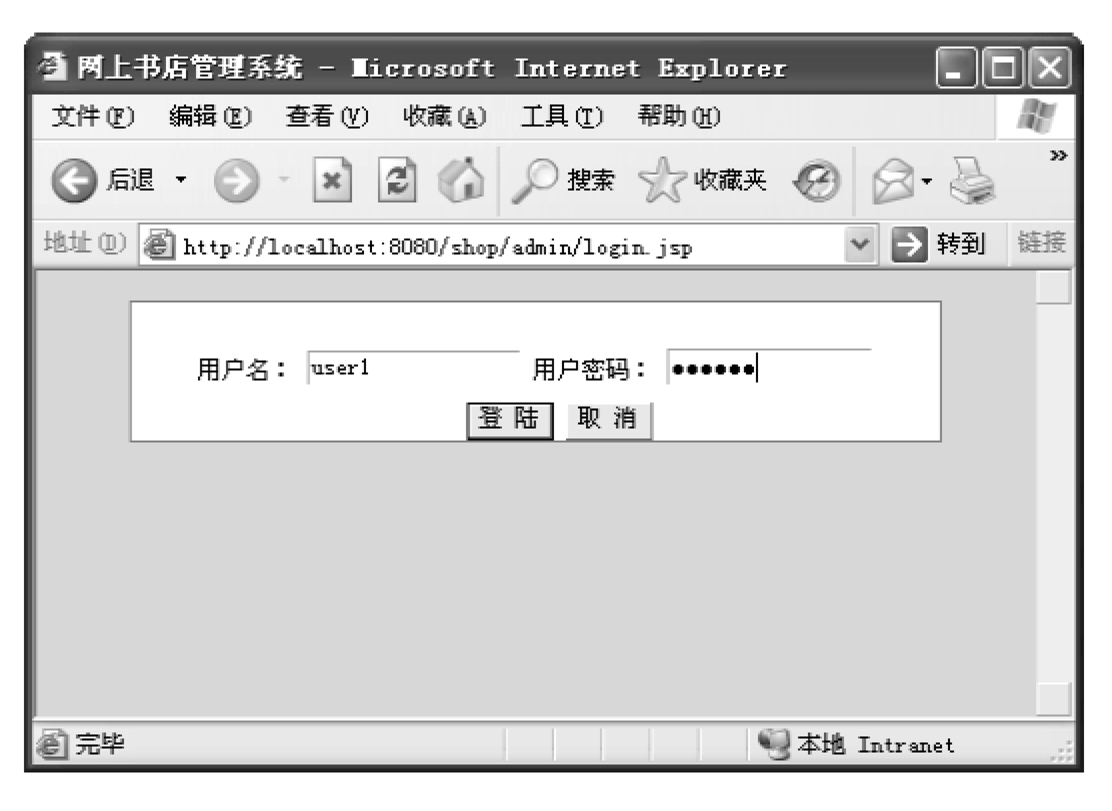


图 13-1　后台管理系统登录页面

（3）在【用户名】文本框中输入“user1”，在【用户密码】文本框中输入“123456”，单击【登录】按钮，进入如图 13-2 所示的后台管理系统。

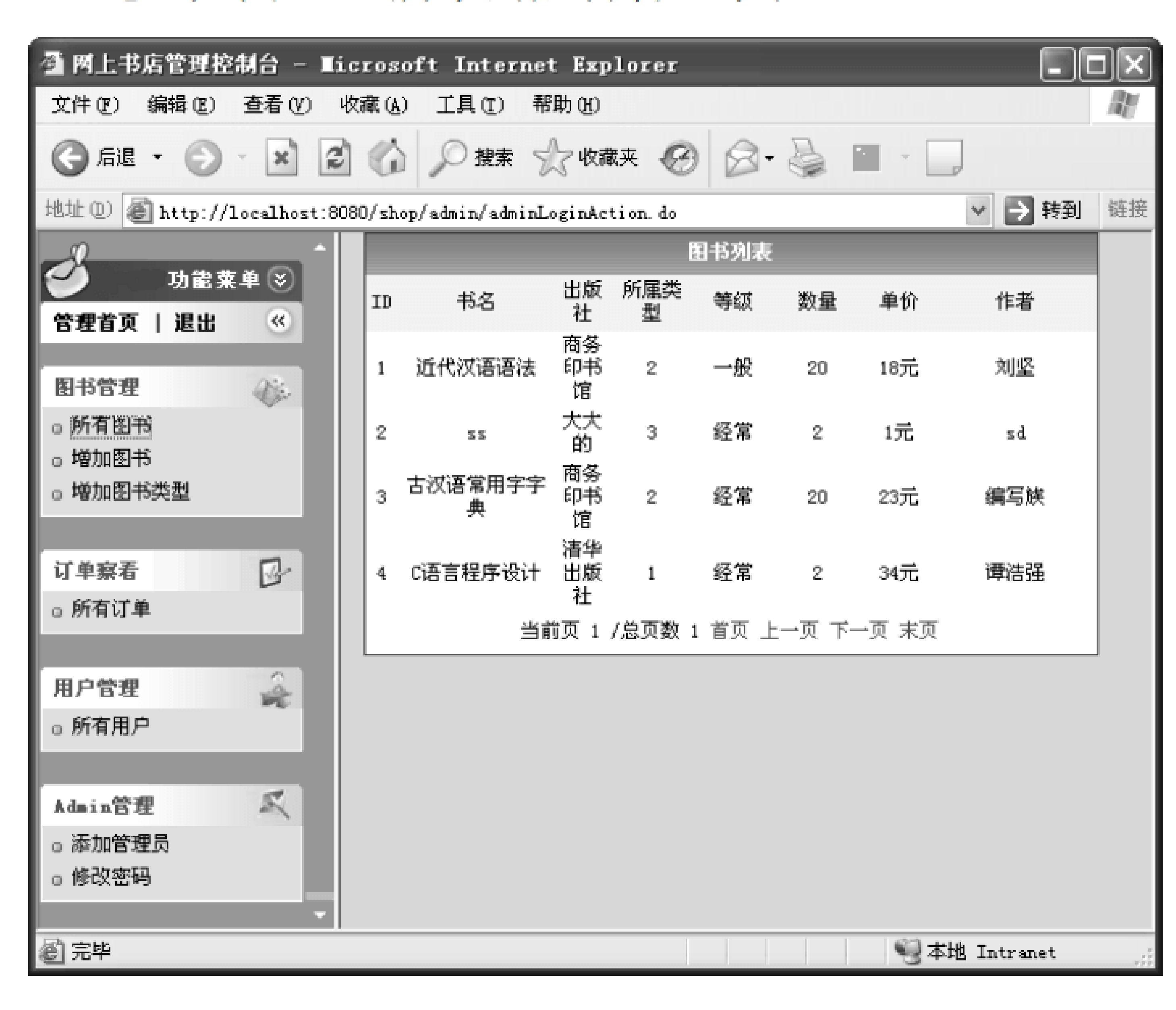


图 13-2　网上书店后台管理系统

（4）选择【所有图书】选项，将列出所有的图书信息，如图 13-3 所示。

图书列表

ID	书名	出版社	所属类型	等级	数量	单价	作者
1	近代汉语语法	商务印书馆	2	一般	20	18元	刘坚
2	ss	大大的	3	经常	2	1元	sd
3	古汉语常用字字典	商务印书馆	2	经常	20	23元	编写族
4	C语言程序设计	清华出版社	1	经常	2	34元	谭浩强

当前页 1 /总页数 1 首页 上一页 下一页 末页

图 13-3　图书列表

（5）选择【增加图书】选项，在如图 13-4 所示的页面中输入新增书籍的信息。单击【新增】按钮，将把该条图书信息保存到数据库中。

图书增加

商品名称：			
商品类型：	请选择子类	出版社：	
价　　格：		库存数量	
作　　者：		等级	请选择子类
图　　片:	上传图片路径： 浏览...		
详细介绍：			

新增

图 13-4　图书增加页面

（6）选择【增加图书类型】选项，在【类型名称】文本框中输入要增加的类型名称，比如“饮食”，单击【确定】按钮，就会添加一个名称为“饮食”的商品类型，添加后如图 13-5 所示。


商品类型

编号	费用类型名称	操作
1	计算机	编辑 | 删除
2	文学	编辑 | 删除
3	保健	编辑 | 删除
4	饮食	编辑 | 删除

添加费用类型

类型名称　　**不得超过 10 个汉字

确 定　清 除

图 13-5　新增图书类型

（7）选择【用户管理】下的【所有用户】选项，将显示所有的用户信息，如图 13-6 所示。



图 13-6　用户列表

（8）选择【编辑】选项，对应用户的信息处于如图 13-7 所示的编辑状态，修改用户的相关信息，单击【更新】按钮，即可将更新后的信息保存到数据库中。



图 13-7　更新用户信息

（9）选择【删除】选项，将出现如图 13-8 所示的提示信息，单击【确定】按钮，将该条用户信息从数据库中删除。

（10）选择【添加管理员】选项，进入“添加管理员”页面，如图 13-9 所示。输入用户名、密码和确认密码就可以添加管理员。



图 13-8　确认删除

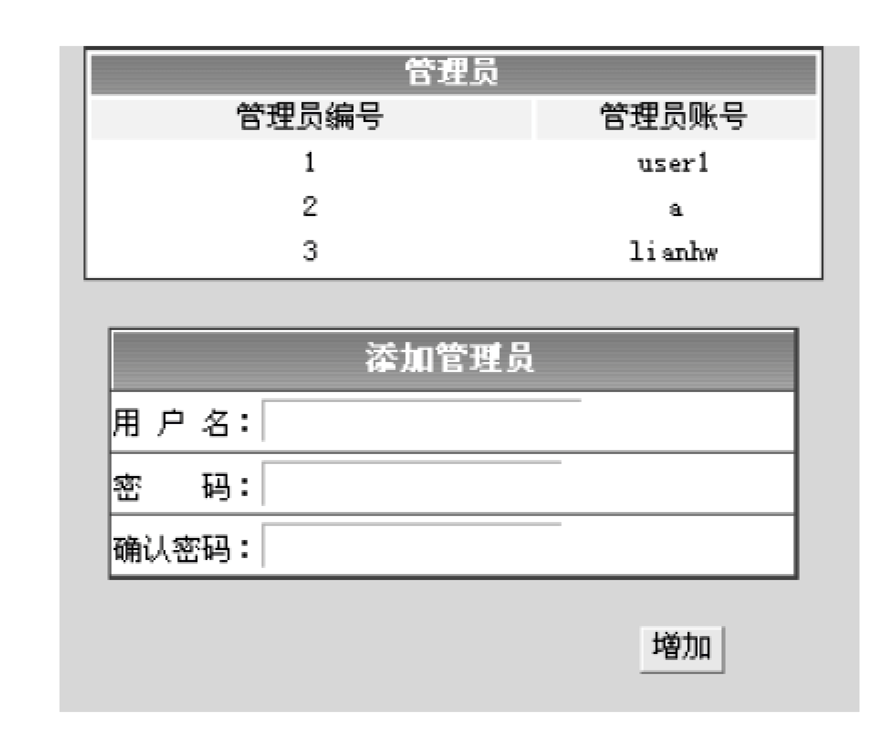


图 13-9　添加管理员

（11）选择【修改密码】选项，进入“修改管理员密码”页面，如图 13-10 所示。输入用户名、原始密码、新密码和确认密码即可修改管理员的密码。



图 13-10　修改管理员密码

（12）打开浏览器，在地址栏中输入 http://localhost:8080/shop/jsp/index.jsp，进入网上书店前台展示系统。其包括如下几个功能模块：图书的分类显示，如图 13-11 所示；用户登录，如图 13-12 所示。

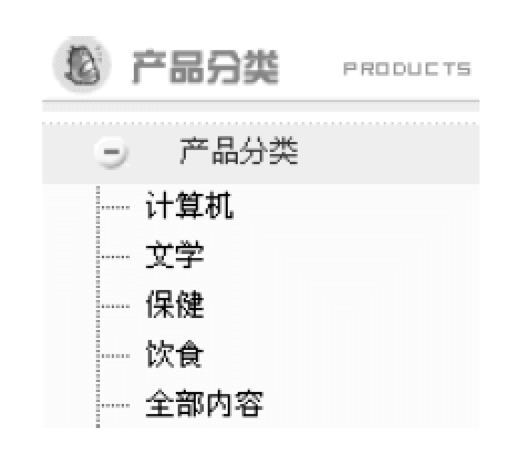


图 13-11　图书分类功能



图 13-12　会员登录系统

13.3　数据库的设计

网上书店管理应用系统后台的图书信息和用户信息都保存在 SQL Server 数据库中，数据库的设计是系统设计的主要方面，本章介绍网上书店系统数据库的设计。

跟我做

（1）打开 SQL Server 企业管理器，创建名称为“MyShop”的数据库，如图 13-13 所示。



图 13-13　MyShop 数据库

（2）打开 SQL Server 的 SQL 查询分析器，选择默认数据库为刚才创建的“MyShop”数据库，输入如下 SQL 脚本：

```
//创建名称为 BookStep 的表
CREATE TABLE [dbo].[BookStep] (
	[BookStepId] [int] IDENTITY (1, 1) NOT NULL ,
```

```
    [Display] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL
) ON [PRIMARY]
GO
//创建名称为 City 的表
CREATE TABLE [dbo].[City] (
    [CityId] [int] IDENTITY (1, 1) NOT NULL ,
    [Display] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL
) ON [PRIMARY]
GO
//创建名称为 Critique 的表
CREATE TABLE [dbo].[Critique] (
    [CritiqueId] [int] IDENTITY (1, 1) NOT NULL ,
    [userId] [int] NOT NULL ,
    [BookId] [int] NOT NULL ,
    [Content] [varchar] (100) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [CritiqueTime] [datetime] NOT NULL
) ON [PRIMARY]
GO
//创建名称为 OrderForm 的表
CREATE TABLE [dbo].[OrderForm] (
    [OrderId] [int] IDENTITY (1, 1) NOT NULL ,
    [UserId] [int] NOT NULL ,
    [BookId] [int] NOT NULL ,
    [amount] [int] NOT NULL ,
    [OrderData] [varchar] (14) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [money] [int] NOT NULL
) ON [PRIMARY]
GO
//创建名称为 Step 的表
CREATE TABLE [dbo].[Step] (
    [StepId] [int] IDENTITY (1, 1) NOT NULL ,
    [Display] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL
) ON [PRIMARY]
GO
//创建名称为 Type 的表
CREATE TABLE [dbo].[Type] (
    [TypeId] [int] IDENTITY (1, 1) NOT NULL ,
    [Display] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL
) ON [PRIMARY]
GO
//创建名称为 admin 的表
CREATE TABLE [dbo].[admin] (
    [userId] [int] IDENTITY (1, 1) NOT NULL ,
    [userName] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
    [UserPwd] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL
) ON [PRIMARY]
GO
//创建名称为 information 的表
CREATE TABLE [dbo].[information] (
```

```
        [BookId] [int] IDENTITY (1, 1) NOT NULL ,
        [BookName] [varchar] (40) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [BookPenster] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [BookCompany] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [BookSynopsis] [varchar] (500) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [BookStorage] [int] NOT NULL ,
        [BookSell] [int] NOT NULL ,
        [BookDate] [datetime] NOT NULL ,
        [BookPrice] [int] NOT NULL ,
        [BookType] [int] NOT NULL ,
        [bookImage] [varchar] (40) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [BookStep] [int] NOT NULL
) ON [PRIMARY]
GO
//创建名称为 users 的表
CREATE TABLE [dbo].[users] (
        [userId] [int] IDENTITY (1, 1) NOT NULL ,
        [userName] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [UserPwd] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [realName] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [UserSex] [int] NOT NULL ,
        [UserPhone] [varchar] (16) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [UserMail] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [UserCity] [int] NOT NULL ,
        [UserAdds] [varchar] (40) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [UserCode] [int] NOT NULL ,
        [UserWork] [varchar] (40) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [Usercard] [varchar] (20) COLLATE Chinese_PRC_CI_AS NOT NULL ,
        [UserStep] [int] NOT NULL ,
        [UserAge] [int] NOT NULL
) ON [PRIMARY]
GO
```

上述 SQL 语句在数据库中创建了 BookStep、City、Critique、OrderForm、Step、Type、admin、information 和 users 表，用来保存网上书店管理应用系统的所有信息。

（3）在查询分析器中输入如下 SQL 语句，创建分页显示用户信息的存储过程。

```
//创建分页显示用户信息的存储过程
CREATE procedure proc_users_page
@curPage int=1,
@perPageRecords int,
@countpage int output
as
declare @total_count int
select @total_count=count(*) from users
declare @page_in_show int
set @page_in_show=@total_count-(@curPage-1)*@perPageRecords
set @countpage=(@total_count+@perPageRecords-1)/@perPageRecords
exec('select top '+@perPageRecords+' * from (select top '+@page_in_show+' * from users order
by userId desc) as a order by userId asc')
```

该存储过程从 users 表中查询到所有的用户信息，将其分页显示。

（4）在查询分析器中输入如下 SQL 语句，定义分页显示图书信息的存储过程。

```
//定义分页显示图书信息的存储过程

CREATE proc proc_page
@curPage int=1,
@perPageRecords int,
@type int,
@step int,
@countpage int output
as

declare @total_count int --总记录数
declare @temp int
set @temp=0
if @type<>0
begin
      if @step<>0
            select @total_count=count(*) from information where bookType=@type and bookStep=
@step
      else
            select @total_count=count(*) from information where bookType=@type
end else
begin
      if @step<>0
            select @total_count=count(*) from information where bookStep= @step
      else
            select @total_count=count(*) from information
end

set @temp=@total_count/@perPageRecords

if @total_count % @perPageRecords>0
begin
      set @countpage=@temp+1
end else
begin
      set @countpage=@temp
end
declare @page_in_show int
set @page_in_show=@total_count-(@curPage-1)*@perPageRecords
if @type=0
begin
      if @step=0
            exec('select top '+@perPageRecords+' * from (select top '+@page_in_show+' * from
information order by bookId desc) as a order by bookId asc')
      else
            exec('select top '+@perPageRecords+' * from (select top '+@page_in_show+' * from
```

```
information where bookStep='+@step+'   order by bookId desc) as a order by bookId asc')
end else
begin
     if @step=0
          exec('select top '+@perPageRecords+' * from (select top '+@page_in_show+' * from
information where bookType='+@Type+'   order by bookId desc) as a order by bookId asc')
     else
          exec('select top '+@perPageRecords+' * from (select top '+@page_in_show+' * from
information where bookType='+@Type+'   and bookStep='+@step+' order by bookId desc) as a
order by bookId asc')
end
```

该存储过程从 information 表中查询到所有的图书信息，并且采用分页的方式将其显示出来。

（5）在查询分析器中输入如下 SQL 语句，定义分页显示订单信息的存储过程。

```
CREATE procedure proc_order_page
@curPage int=1,
@perPageRecords int,
@countpage int output
as
declare @total_count int
select @total_count=count(*) from OrderForm
declare @page_in_show int
set @page_in_show=@total_count-(@curPage-1)*@perPageRecords
set @countpage=(@total_count+@perPageRecords-1)/@perPageRecords
exec('select top '+@perPageRecords+' * from (select top '+@page_in_show+' * from OrderForm
order by UserId desc) as a order by UserId asc')
```

该存储过程从 OrderForm 表中查询到所有的订单信息，并且采用分页的方式将其显示出来。

（6）单击按钮，执行上述 SQL 脚本，生成了几张表，如图 13-14 所示。

表　9 个项目

名称	所有者	类型	创建日期
admin	dbo	用户	2006-10-1 9:30:54
BookStep	dbo	用户	2006-10-1 9:30:54
City	dbo	用户	2006-10-1 9:30:54
Critique	dbo	用户	2006-10-1 9:30:54
information	dbo	用户	2006-10-1 9:30:54
OrderForm	dbo	用户	2006-10-1 9:30:54
Step	dbo	用户	2006-10-1 9:30:54
Type	dbo	用户	2006-10-1 9:30:54
users	dbo	用户	2006-10-1 9:30:54

图 13-14　MyShop 中的所有表

13.4　系统数据库操作的封装

网上书店系统用数据库存储所有与系统有关的信息，对数据库操作是系统中的常用操作。本节将所有与数据库相关联的操作进行封装，实现代码的模块化。通过编写模块化代

码可以提高软件可移植性，同时会使整个系统的代码结构变得非常清晰。

13.4.1　对后台管理系统的数据库操作

本小节实现对网上书店管理应用系统后台管理系统的数据库操作，包括验证用户的登录信息、分页返回所有的用户等操作。

系统采用 JDBC 访问数据库，SQL Server 数据库的驱动程序采用 com.microsoft.jdbc.sqlserver.SQLServerDriver。在实际的应用中对于 SQL Server 数据库最好采用 JDBC-ODBC 桥的方式，因为不同的数据库驱动程序在性能方面差距较大，采用 JDBC-ODBC 桥的方式是一种理想的方式。

跟我做

（1）创建名称为“shop”的 Java 工程。在 shop 工程中建立名称为“src”的源文件夹，在其中建立名称为“db”的包，并创建 AdminOperation.java 类。

（2）编辑 AdminOperation 类，输入如下代码，配置数据库的连接信息，建立与数据库的连接。

```
/**
 * 配置数据库的连接信息，建立与数据库的连接
 */
public AdminOperation() {
	try {
		// 加载数据库的驱动程序类
		Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
		// 驱动与数据库的连接
		connection = DriverManager
				.getConnection(

						"jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=My
						Shop","sa", "");
	} catch (SQLException ex) {

	} catch (ClassNotFoundException ex) {

	}

}
```

与数据库的连接是通过 JDBC 实现的，需要加载数据库的驱动程序类，并配置数据库的 URL、用户名和用户密码等信息。

（3）编辑 AdminOperation 类，输入如下代码，验证用户的登录信息。

```
/**
 * 验证用户的登录信息
 *
```

```
 * @param ab
 * @return
 */
public boolean checkAdminLogin(AdminBean ab) {

	boolean result = false;
	try {
		// 查询 Admin 表的 sql 语句
		String sql = "select count(*) from Admin where UserName=? and UserPwd=?";
		// 创建 PreparedStatement 对象
		preparedStatement = connection.prepareStatement(sql);
		// 设置 sql 语句的第一个参数为用户名
		preparedStatement.setString(1, ab.getUserName());
		// 设置 sql 语句的第二个参数为用户密码
		preparedStatement.setString(2, ab.getUserPwd());
		// 执行查询操作
		resultSet = preparedStatement.executeQuery();
		// 如果查询结果不为空就说明该用户为合法用户
		if (resultSet.next()) {
			if (resultSet.getInt(1) > 0) {
				result = true;
			}
		}
	} catch (Exception e) {
		e.printStackTrace();
	}
	return result;
}
```

参数 AdminBean 中保存着用户名和用户密码，根据这些信息查询数据库中是否存在该用户，从而验证用户的合法性。

（4）编辑 AdminOperation 类，输入如下代码信息，实现分页返回用户信息。

```
/**
 * 分页返回所有的用户
 *
 * @param count
 * @param page
 * @return
 */
public ArrayList getUsersList(int count, int page) {
	ArrayList list = new ArrayList();
	try {
		// 调用存储过程，创建 CallableStatement 对象
		callableStatement = connection
				.prepareCall("{call proc_users_page (?,?,?)}");
		// 设置 callableStatement 对象的第一个参数的值
		callableStatement.setInt(1, page);
		// 设置 callableStatement 对象的第二个参数的值
		callableStatement.setInt(2, count);
```

```
                // 设置 callableStatement 对象输出值为整形
                callableStatement.registerOutParameter(3, Types.INTEGER);
                // 执行查询
                resultSet = callableStatement.executeQuery();
                // 将查询到的结果组装成 UserBean 对象放到 list 中
                while (resultSet.next()) {
                    UserBean ub = new UserBean();
                    ub.setUserId(resultSet.getInt(1));
                    ub.setUserName(resultSet.getString(2));
                    ub.setUserPwd(resultSet.getString(3));
                    ub.setRealName(resultSet.getString(4));
                    ub.setUserSex(resultSet.getInt(5));
                    ub.setUserPhone(resultSet.getString(6));
                    ub.setUserMail(resultSet.getString(7));
                    ub.setUserCity(resultSet.getInt(8));
                    ub.setUserAdds(resultSet.getString(9));
                    ub.setUserCode(resultSet.getInt(10));
                    ub.setUserWork(resultSet.getString(11));
                    ub.setUserCard(resultSet.getString(12));
                    ub.setUserStrp(resultSet.getInt(13));
                    ub.setUserAge(resultSet.getInt(14));
                    list.add(ub);
                }
                pagecount = callableStatement.getInt(3);
            } catch (SQLException ex) {
                System.out.println("服务器异常发生在 getUsersList()");
                ex.printStackTrace();
            }
            return list;
        }
```

（5）编辑 AdminOperation 类，输入如下代码信息，实现插入新的管理员信息。

```
        /**
         * 插入新的管理员信息
         *
         * @param admin
         * @return
         */
        public boolean insertAdmin(AdminBean admin) {
            try {
                // 创建插入管理员信息的 PreparedStatement 对象
                preparedStatement = connection
                        .prepareStatement("insert into admin (userName,UserPwd) values(?,?) ");
                // 设置 preparedStatement 对象的第一个参数
                preparedStatement.setString(1, admin.getUserName());
                // 设置 preparedStatement 对象的第二个参数
                preparedStatement.setString(2, admin.getUserPwd());
                // 执行更新操作
                int flag = preparedStatement.executeUpdate();
```

```
                if (flag == 0) {
                        return false;
                }
        } catch (SQLException ex) {
                ex.printStackTrace();
                return false;
        }
        return true;

}
```

该方法将用户提交的管理员信息插入到数据库中的 admin 表中，从而创建新的管理员用户。

（6）编辑 AdminOperation 类，输入如下代码信息，从而取得用户的等级信息。

```
/**
 * 取得等级信息
 *
 * @param stepId
 * @return
 */
public String getStep(int stepId) {
        try {
                // 查询 Step 信息的 preparedStatement 语句
                preparedStatement = connection
                                .prepareStatement("select Display from Step where StepId=?");
                // 设置 perparedStatement 语句的第一个参数
                preparedStatement.setInt(1, stepId);
                // 执行查询
                resultSet = preparedStatement.executeQuery();
                // 将结果集指针后移
                resultSet.next();
                // 取得 display 字段的值
                return resultSet.getString("Display");
        } catch (SQLException ex) {

                return null;
        }

}
```

该方法通过给定的 stepid，取得相应的用户级别信息。

（7）编辑 AdminOperation 类，输入如下代码信息，取得书籍的等级信息。

```
/**
 * 取得书籍等级
 *
 * @param stepId
 * @return
 */
```

```
public String getBookStep(int stepId) {
	try {
		// 创建从 BookStep 取得 BookStep 信息的 Statement 对象
		preparedStatement = connection
				.prepareStatement("select Display from BookStep where BookStepId =?");
		// 设置 preparedStatement 对象的第一个参数
		preparedStatement.setInt(1, stepId);
		// 执行数据库查询操作
		ResultSet res = preparedStatement.executeQuery();
		// 将结果集指针后移
		res.next();
		// 取得结果集中的 display 字段的值
		return res.getString("Display");
	} catch (SQLException ex) {

		return null;
	}

}
```

该方法通过给定的 stepid，取得相应的书籍级别信息。

（8）编辑 AdminOperation 类，输入如下代码信息，实现插入 BookType 信息。

```
/**
 * 插入 BookType 信息
 *
 * @param btb
 * @return
 */
public boolean insertBookType(BookTypeBean btb) {
	try {
		// 往 Type 表中插入一条记录的 Statement 对象
		preparedStatement = connection
				.prepareStatement("insert into Type (Display) values('"
						+ btb.getDisplay() + "')");
		// 执行更新操作
		int flag = preparedStatement.executeUpdate();
		if (flag == 0) {
			return false;
		}
	} catch (SQLException ex) {
		ex.printStackTrace();
		return false;
	}
	return true;
}
```

该方法将新的图书类别信息插入到 Type 表中。

（9）编辑 AdminOperation 类，输入如下代码信息，删除数据库中的 BookType 信息。

```
/**
 * 删除 BookType 信息
 *
 * @param BookTypeId
 * @return
 */
public boolean deleteBookType(int BookTypeId) {
	try {
		// 创建从 Type 表中删除记录的 Statement 对象
		preparedStatement = connection
				.prepareStatement("delete Type where TypeId=?");
		// 设置 Statement 对象的第一个参数
		preparedStatement.setInt(1, BookTypeId);
		// 执行删除操作
		int flag = preparedStatement.executeUpdate();
		if (flag == 0) {
			return false;
		}
	} catch (SQLException ex) {
		ex.printStackTrace();
		return false;
	}
	return true;
}
```

该方法从数据库中删除 BookType 信息。

（10）编辑 AdminOperation 类，输入如下代码信息，实现更新 BookType 信息操作。

```
/**
 * 更新 BookType 信息
 *
 * @param btb
 * @return
 */
public boolean updateBookType(BookTypeBean btb) {
	try {
		// 创建在 Type 表中更新 BookType 记录的 Statement 对象
		preparedStatement = connection
				.prepareStatement("update Type set Display=? where TypeId=?");
		// 设置 Statement 对象的第一个参数
		preparedStatement.setString(1, btb.getDisplay());
		// 设置 Statement 对象的第二个参数
		preparedStatement.setInt(2, btb.getTypeId());
		// 执行更新操作
		int flag = preparedStatement.executeUpdate();
		if (flag == 0) {
			return false;
		}
```

```
            } catch (SQLException ex) {
                ex.printStackTrace();
                return false;
            }
            return true;
        }
```

该方法实现了更新 BookType 信息的操作。

（11）编辑 AdminOperation 类，输入如下代码信息，实现增加图书信息。

```
    /**
     * 增加图书信息
     *
     * @param bb
     * @return
     */
    public boolean insertBooks(BookBean bb) {
        try {
            // 创建插入图书信息的 Statement 对象
            preparedStatement = connection
                    .prepareStatement("insert into information (BookName,BookPenster,
BookCompany,BookSynopsis,BookStorage,BookSell,BookDate,BookPrice,BookType,bookImage,
BookStep) values(?,?,?,?,?,?,?,?,?,?,?)");
            // 设置 Statement 对象的第一个参数
            preparedStatement.setString(1, bb.getBookName());
            // 设置 Statement 对象的第二个参数
            preparedStatement.setString(2, bb.getBookPenster());
            // 设置 Statement 对象的第三个参数
            preparedStatement.setString(3, bb.getBookCompany());
            // 设置 Statement 对象的第四个参数
            preparedStatement.setString(4, bb.getBookSynopsis());
            // 设置 Statement 对象的第五个参数
            preparedStatement.setInt(5, bb.getBookStorage());
            // 设置 Statement 对象的第六个参数
            preparedStatement.setInt(6, bb.getBookSell());
            // 设置 Statement 对象的第七个参数
            preparedStatement.setDate(7, new java.sql.Date(new java.util.Date()
                    .getTime()));
            // 设置 Statement 对象的第八个参数
            preparedStatement.setInt(8, bb.getBookPrice());
            // 设置 Statement 对象的第九个参数
            preparedStatement.setInt(9, Integer.parseInt(bb.getBookType()));
            // 设置 Statement 对象的第十个参数
            preparedStatement.setString(10, bb.getBookImage());
            // 设置 Statement 对象的第十一个参数
            preparedStatement.setInt(11, bb.getBookStep());
            // 执行更新操作
            int flag = preparedStatement.executeUpdate();
            if (flag == 0) {
                return false;
```

```
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
                return false;
            }
            return true;
        }
```

（12）编辑 AdminOperation 类，输入如下代码，实现从数据库中取得某条图书信息的操作。

```
    /**
     * 取得某条图书信息
     *
     * @param bookId
     * @return
     */
    public BookBean getBookBean(int bookId) {
        BookBean bbean = new BookBean();
        // 查询某条图书信息的 SQL 语句
        String sql = "select * from information where BookId=?";
        try {
            // 创建 preparedStatement 对象
            preparedStatement = connection.prepareStatement(sql);
            // 设置 Statement 对象的第一个参数的值
            preparedStatement.setInt(1, bookId);
            // 执行查询
            resultSet = preparedStatement.executeQuery();
            // 根据查询结果封装成 BookBean 对象
            while (resultSet.next()) {
                bbean.setBookCompany(resultSet.getString("BookCompany"));
                bbean.setBookCount(1);
                bbean.setBookData(resultSet.getDate("BookDate"));
                bbean.setBookId(bookId);
                bbean.setBookImage(resultSet.getString("BookImage"));
                bbean.setBookName(resultSet.getString("BookName"));
                bbean.setBookPenster(resultSet.getString("BookPenster"));
                bbean.setBookPrice(resultSet.getInt("BookPrice"));
                bbean.setBookSell(resultSet.getInt("BookSell"));
                bbean.setBookStep(resultSet.getInt("BookStep"));
                bbean.setBookStorage(resultSet.getInt("BookStorage"));
                bbean.setBookSynopsis(resultSet.getString("BookSynopsis"));
                bbean.setBookType(this.queryType(""
                        + resultSet.getInt("BookType")));
                bbean.setBookAllPrice(bbean.getBookPrice());
            }
        }

        catch (SQLException ex) {
            ex.printStackTrace();
```

```
                return null;
            }
            return bbean;
    }
```

（13）编辑 AdminOperation 类，输入如下代码，实现根据 typeID 查询其对应的类型值信息。

```
    /**
     * 根据 typeID 查询其对应的类型值
     *
     * @param typeId
     * @return
     */
    public String queryType(String typeId) {
            try {
                    // 创建相应的 Statement 对象
                    java.sql.PreparedStatement prepar1 = connection
                                    .prepareStatement("select Display from Type where TypeId=?");
                    // 设置 Statement 对象的第一个参数
                    prepar1.setInt(1, Integer.parseInt(typeId));
                    // 执行查询取得查询结果
                    ResultSet res1 = prepar1.executeQuery();
                    res1.next();
                    return res1.getString("Display");
            } catch (SQLException ex) {

                    return null;
            }
    }
```

（14）编辑 AdminOperation 类，输入如下代码，实现从数据库中读取所有书籍信息的操作。

```
    /**
     * 返回书籍信息列表
     *
     * @param count
     * @param page
     * @return
     */
    public ArrayList getBooksList(int count, int page) {
            ArrayList list = new ArrayList();
            try {
                    // 调用查询书籍的存储过程
                    callableStatement = connection
                                    .prepareCall("{call proc_books_page (?,?,?)}");
                    // 设置 Statement 对象的相关参数
                    callableStatement.setInt(1, page);
                    callableStatement.setInt(2, count);
```

```
                // 设置 Statement 对象的输入参数类型
                callableStatement.registerOutParameter(3, Types.INTEGER);
                // 执行查询
                resultSet = callableStatement.executeQuery();
                // 将查询结果放置到一个链表中保存
                while (resultSet.next()) {
                        BookBean kb = new BookBean();
                        kb.setBookId(resultSet.getInt(1));
                        kb.setBookName(resultSet.getString(2));
                        kb.setBookPenster(resultSet.getString(3));
                        kb.setBookCompany(resultSet.getString(4));
                        kb.setBookSynopsis(resultSet.getString(5));
                        kb.setBookStorage(resultSet.getInt(6));
                        kb.setBookSell(resultSet.getInt(7));
                        kb.setBookData(resultSet.getDate(8));
                        kb.setBookPrice(resultSet.getInt(9));
                        kb.setBookType(resultSet.getInt(10) + "");
                        kb.setBookImage(resultSet.getString(11));
                        kb.setBookStep(resultSet.getInt(12));
                        list.add(kb);
                }
                pagecount = callableStatement.getInt(3);
        } catch (SQLException ex) {
                System.out.println("服务器异常发生在 getBooksList()");
                ex.printStackTrace();
        }
        return list;
}
```

（15）编辑 AdminOperation 类，输入如下代码，属性从数据库中取得所有订单信息列表的操作。

```
/**
 * 取得订单列表
 *
 * @param count
 * @param page
 * @return
 */
public ArrayList getOrderList(int count, int page) {
        ArrayList list = new ArrayList();
        try {
                // 调用查询所有订单的存储过程
                callableStatement = connection
                                .prepareCall("{call proc_order_page (?,?,?)}");
                // 设置存储过程的相应参数
                callableStatement.setInt(1, page);
                callableStatement.setInt(2, count);
                // 设置存储过程输出参数的类型
                callableStatement.registerOutParameter(3, Types.INTEGER);
```

```
                // 执行查询
                resultSet = callableStatement.executeQuery();
                // 处理查询结果
                while (resultSet.next()) {
                        OrderFormBean ob = new OrderFormBean();
                        ob.setOrderId(resultSet.getInt(1));
                        ob.setUserId(resultSet.getInt(2));
                        ob.setBookId(resultSet.getInt(3));
                        ob.setAmount(resultSet.getInt(4));
                        ob.setOrderData(resultSet.getString(5));
                        ob.setMoney(resultSet.getInt(6));
                        list.add(ob);
                }
                pagecount = callableStatement.getInt(3);
        } catch (SQLException ex) {
                System.out.println("服务器异常发生在 getBooksList()");
                ex.printStackTrace();
        }
        return list;
}
```

（16）编辑 AdminOperation 类，输入如下代码，实现修改 Admin 密码信息的操作。

```
/**
 * 改变 Admin 的密码信息
 *
 * @param admin
 * @return
 */
public boolean changeAdminPwd(AdminBean admin) {
        try {
                // 创建更新 admin 密码的 Statement 对象
                preparedStatement = connection
                                .prepareStatement("update admin set UserPwd=? where userId=?");
                // 设置其相应的参数
                preparedStatement.setString(1, admin.getUserPwd());
                preparedStatement.setInt(2, admin.getUserId());
                // 执行更新操作
                int flag = preparedStatement.executeUpdate();
                if (flag == 0) {
                        return false;
                }
        } catch (SQLException ex) {
                ex.printStackTrace();
                return false;
        }
        return true;
}
```

（17）编辑 AdminOperation 类，输入如下代码，实现关闭与数据库的所有连接的操作。

```
        /**
         * 关闭与数据库的所有连接
         */
        public void Close() {
                try {
                        if (resultSet != null) {
                                // 关闭 ResultSet 对象
                                resultSet.close();
                        }
                        if (preparedStatement != null) {
                                // 关闭 PreparedStatement 对象
                                preparedStatement.close();
                        }
                        if (connection != null) {
                                // 关闭 Connection 对象
                                connection.close();
                        }
                } catch (SQLException ex) {
                        ex.printStackTrace();
                }

        }
}
```

在 AdminOperation 类的构造函数中配置了与数据库的连接信息，在后台管理模块中涉及的所有对数据库的查询操作、删除操作和更新操作都在这个类中实现。将对数据库的所有操作进行统一封装，有利于代码的维护和模块化。

13.4.2　对前台展示系统的数据库操作

本小节实现网上书店管理应用系统前台展示系统的所有数据库操作，包括验证登录用户的合法性、根据用户名查询用户信息和根据 cityid 查询城市信息等操作。

系统通过 JDBC 标准化接口实现对数据库的访问，首先通过数据库的 URL、用户名和密码等信息建立与数据库的连接，在系统运行期间为了提高系统的性能，将始终保持与数据库的连接完成查询、插入、更新等数据库操作。

跟我做

（1）在“db”包中创建 CustomerOperation.java 类，编辑该文件，输入如下代码信息，构造与数据库的连接信息。

```
        /**
         * 构造函数，构造与数据库的连接信息
         */
        public CustomerOperation() {
                try {
```

```
                // 加载 SQL Server 的数据库驱动程序类
                Class.forName("com.microsoft.jdbc.sqlserver.SQLServerDriver");
                // 创建 Connection 对象
                connection = DriverManager
                        .getConnection(
                                "jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=My
                                Shop", "sa", "");
        } catch (SQLException ex) {

        } catch (ClassNotFoundException ex) {

        }

    }
```

与数据库的连接是通过 JDBC 实现的，需要加载数据库的驱动程序类，并配置数据库的 URL、用户名和用户密码等信息。

（2）编辑 CustomerOperation 类，输入如下代码，实现验证登录用户合法性的操作。

```
    /**
     * 验证登录用户的合法性
     *
     * @param userName
     * @param userPwd
     * @return
     */
    public UserBean checkUsersLogin(String userName, String userPwd) {
        UserBean useBean;
        flag = false;
        try {
            // 从 users 表中查询特定用户的 SQL 语句
            String sql = "select count(*) from users where userName=? and userPwd=?";
            // 创建 PreparedStatement 对象
            preparedStatement = connection.prepareStatement(sql);
            // 为 PreparedStatement 对象设置必要的参数
            preparedStatement.setString(1, userName);
            preparedStatement.setString(2, userPwd);
            // 执行数据库查询操作
            resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                // 判断查询结果是否为空
                if (resultSet.getInt(1) > 0) {
                    useBean = this.getUser(userName);
                } else {
                    useBean = null;
                }
            } else {
                useBean = null;
            }
        } catch (Exception e) {
```

```
				useBean = null;
				e.printStackTrace();
			}
			return useBean;
		}
```

（3）编辑 CustomerOperation 类，输入如下代码，根据用户名查询其全部的用户信息。

```
		/**
		 * 根据用户名查询其全部的用户信息
		 *
		 * @param userName
		 * @return
		 */
		public UserBean getUser(String userName) {
			UserBean useBean = new UserBean();
			// 查询特定用户信息的 SQL 语句
			String sql = "select * from users where userName=?";
			try {
				// 创建 Statement 对象
				preparedStatement = connection.prepareStatement(sql);
				// 设置 Statement 对象的相关参数
				preparedStatement.setString(1, userName);
				// 执行查询语句
				resultSet = preparedStatement.executeQuery();
				// 对查询到的结果集进行处理
				while (resultSet.next()) {
					useBean.setRealName(resultSet.getString("realName"));
					useBean.setUserAdds(resultSet.getString("UserAdds"));
					useBean.setUserAge(resultSet.getInt("UserAge"));
					useBean.setUserCard(resultSet.getString("Usercard"));
					useBean.setUserCity(resultSet.getInt("UserCity"));
					useBean.setUserCode(resultSet.getInt("UserCode"));
					useBean.setUserId(resultSet.getInt("userId"));
					useBean.setUserMail(resultSet.getString("UserMail"));
					useBean.setUserName(resultSet.getString("userName"));
					useBean.setUserPhone(resultSet.getString("UserPhone"));
					useBean.setUserPwd(resultSet.getString("UserPwd"));
					useBean.setUserSex(resultSet.getInt("UserSex"));
					useBean.setUserStrp(resultSet.getInt("UserStep"));
					useBean.setUserWork(resultSet.getString("UserWork"));
				}
			} catch (SQLException ex) {
				ex.printStackTrace();
			}

			return useBean;
		}
```

（4）编辑 CustomerOperation 类，输入如下代码，根据 cityId 信息查询城市名。

```
/**
 * 根据 cityId 信息查询城市名
 *
 * @param cityId
 * @return
 */
public String getCity(int cityId) {
	try {
		// 创建 Statement 对象
		preparedStatement = connection
				.prepareStatement("select Display from City where CityId=?");
		// 设置其对应的参数
		preparedStatement.setInt(1, cityId);
		// 执行查询
		resultSet = preparedStatement.executeQuery();
		resultSet.next();
		return resultSet.getString("Display");
	} catch (SQLException ex) {

		return null;
	}

}
```

（5）编辑 CustomerOperation 类，输入如下代码，根据 stepId 取得相应的等级信息。

```
/**
 * 根据 stepId 取得相应等级信息
 *
 * @param stepId
 * @return
 */
public String getStep(int stepId) {
	try {
		// 创建 Statement 对象
		preparedStatement = connection
				.prepareStatement("select Display from Step where StepId=?");
		// 设置其对应的参数
		preparedStatement.setInt(1, stepId);
		// 执行查询
		resultSet = preparedStatement.executeQuery();
		resultSet.next();
		return resultSet.getString("Display");
	} catch (SQLException ex) {

		return null;
	}

}
```

（6）编辑 CustomerOperation 类，输入如下代码，根据 userId 来查询其对应的订单编号。

```
public int getUserOrderCount(int userId) {
	// 查询某个用户订单数量的查询语句
	String sql = "select count(*) from OrderForm where userId=?";
	try {
		// 创建 Statement 对象
		preparedStatement = connection.prepareStatement(sql);
		// 设置其对应的参数
		preparedStatement.setInt(1, userId);
		// 执行查询
		resultSet = preparedStatement.executeQuery();
		resultSet.next();
		return resultSet.getInt(1);
	} catch (SQLException ex) {
		ex.printStackTrace();
	}
	return 0;
}
```

（7）编辑 CustomerOperation 类，输入如下代码，根据 userId 取得其用户信息。

```
/**
 * 根据 userId 取得其用户信息
 *
 * @param userId
 * @return
 */
public UserBean getUser(int userId) {
	UserBean useBean = new UserBean();
	// 查询特定用户信息的 sql 语句
	String sql = "select * from users where userId=?";
	try {
		// 创建 Statement 对象
		preparedStatement = connection.prepareStatement(sql);
		// 设置 Statement 对象相应的参数
		preparedStatement.setInt(1, userId);
		// 执行查询
		resultSet = preparedStatement.executeQuery();
		// 对查询结果的处理
		while (resultSet.next()) {
			useBean.setRealName(resultSet.getString("realName"));
			useBean.setUserAdds(resultSet.getString("UserAdds"));
			useBean.setUserAge(resultSet.getInt("UserAge"));
			useBean.setUserCard(resultSet.getString("Usercard"));
			useBean.setUserCity(resultSet.getInt("UserCity"));
			useBean.setUserCode(resultSet.getInt("UserCode"));
			useBean.setUserId(resultSet.getInt("userId"));
			useBean.setUserMail(resultSet.getString("UserMail"));
			useBean.setUserName(resultSet.getString("userName"));
```

```
				useBean.setUserPhone(resultSet.getString("UserPhone"));
				useBean.setUserPwd(resultSet.getString("UserPwd"));
				useBean.setUserSex(resultSet.getInt("UserSex"));
				useBean.setUserStrp(resultSet.getInt("UserStep"));
				useBean.setUserWork(resultSet.getString("UserWork"));
			}
		} catch (SQLException ex) {
			ex.printStackTrace();
		}

		return useBean;
	}
```

（8）编辑 CustomerOperation 类，输入如下代码，实现插入新用户信息的操作。

```
	/**
	 * 插入新用户操作
	 *
	 * @param useBean
	 * @return
	 */
	public boolean insertUser(UserBean useBean) {
		boolean bool = false;
		// 判断该用户是否已经存在
		if (!this.hasUser(useBean.getUserName())) {

			bool = true;
			// 插入新用户的 SQL 语句
			String sql = "insert into users (userName,UserPwd,realName,UserSex,UserPhone,
UserMail,UserCity,UserAdds,UserCode,UserWork,Usercard,UserAge) values(?,?,?,?,?,?,?,?,?,?,?,?)";
			try {
				// 创建 Statement 对象
				preparedStatement = connection.prepareStatement(sql);
				// 设置 Statement 对象的相应参数
				preparedStatement.setString(1, useBean.getUserName());
				preparedStatement.setString(2, useBean.getUserPwd());
				preparedStatement.setString(3, useBean.getRealName());
				preparedStatement.setInt(4, useBean.getUserSex());
				preparedStatement.setString(5, useBean.getUserPhone());
				preparedStatement.setString(6, useBean.getUserMail());
				preparedStatement.setInt(7, useBean.getUserCity());
				preparedStatement.setString(8, useBean.getUserAdds());
				preparedStatement.setInt(9, useBean.getUserCode());
				preparedStatement.setString(10, useBean.getUserWork());
				preparedStatement.setString(11, useBean.getUserCard());
				preparedStatement.setInt(12, useBean.getUserAge());
				// 执行更新操作
				int flag = preparedStatement.executeUpdate();
				if (flag == 0) {
					bool = false;
```

```
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
                bool = false;
            }
        }
        return bool;
    }
```

（9）编辑 CustomerOperation 类，输入如下代码，实现更新特定用户信息的操作。

```
    /**
     * 更新特定用户信息
     *
     * @param useBean
     * @return
     */
    public boolean updateUser(UserBean useBean) {
        boolean bool = false;

        bool = true;
        // 更新用户信息的 sql 语句
        String sql = "update users set userName=?,UserPwd=?,realName=?,UserSex=?, User
Phone=?,UserMail=?,UserCity=?,UserAdds=?,UserCode=?,UserWork=?,Usercard=?,UserAge=?
where userId=?";
        try {
            // 创建 Statement 对象
            preparedStatement = connection.prepareStatement(sql);
            // 设置其相应参数
            preparedStatement.setString(1, useBean.getUserName());
            preparedStatement.setString(2, useBean.getUserPwd());
            preparedStatement.setString(3, useBean.getRealName());
            preparedStatement.setInt(4, useBean.getUserSex());
            preparedStatement.setString(5, useBean.getUserPhone());
            preparedStatement.setString(6, useBean.getUserMail());
            preparedStatement.setInt(7, useBean.getUserCity());
            preparedStatement.setString(8, useBean.getUserAdds());
            preparedStatement.setInt(9, useBean.getUserCode());
            preparedStatement.setString(10, useBean.getUserWork());
            preparedStatement.setString(11, useBean.getUserCard());
            preparedStatement.setInt(12, useBean.getUserAge());
            preparedStatement.setInt(13, useBean.getUserId());
            // 执行更新操作
            int flag = preparedStatement.executeUpdate();
            if (flag == 0) {
                bool = false;
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            bool = false;
```

```
        }
        return bool;
    }
```

（10）编辑 CustomerOperation 类，输入如下代码，判断特定用户是否已经存在。

```
    /**
     * 判断特定用户是否已经存在
     *
     * @param userName
     * @return
     */
    public boolean hasUser(String userName) {
        boolean bool = true;
        // 查询特定用户数量的 SQL 语句
        String sql = "select count(*) from users where userName=?";
        try {
            // 创建 Statement 对象
            preparedStatement = connection.prepareStatement(sql);
            // 设置 Statement 对象的相关参数
            preparedStatement.setString(1, userName);
            // 执行查询
            resultSet = preparedStatement.executeQuery();
            resultSet.next();
            int flag = resultSet.getInt(1);
            if (flag > 0) {
                bool = true;
            } else {
                bool = false;
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            bool = true;
        }

        return bool;
    }
```

（11）编辑 CustomerOperation 类，输入如下代码，根据 bookID 取得相应的图书信息。

```
    /**
     * 根据 bookID 取得相应的图书信息
     *
     * @param bookId
     * @return
     */
    public BookBean getBookBean(String bookId) {
        BookBean bbean = new BookBean();
        // 查询图书信息的 SQL 语句
        String sql = "select * from information where BookId=?";
        try {
```

```
            // 创建 Statement 对象
            preparedStatement = connection.prepareStatement(sql);
            // 设置 Statement 对象相应的参数
            preparedStatement.setString(1, bookId);
            // 执行查询操作
            resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                bbean.setBookCompany(resultSet.getString("BookCompany"));
                bbean.setBookCount(1);
                bbean.setBookData(resultSet.getDate("BookDate"));
                bbean.setBookId(Integer.parseInt(bookId));
                bbean.setBookImage(resultSet.getString("BookImage"));
                bbean.setBookName(resultSet.getString("BookName"));
                bbean.setBookPenster(resultSet.getString("BookPenster"));
                bbean.setBookPrice(resultSet.getInt("BookPrice"));
                bbean.setBookSell(resultSet.getInt("BookSell"));
                bbean.setBookStep(resultSet.getInt("BookStep"));
                bbean.setBookStorage(resultSet.getInt("BookStorage"));
                bbean.setBookSynopsis(resultSet.getString("BookSynopsis"));
                bbean.setBookType(this.queryType(""
                        + resultSet.getInt("BookType")));
                bbean.setBookAllPrice(bbean.getBookPrice());
            }
        } catch (SQLException ex) {
            ex.printStackTrace();
            return null;
        }
        return bbean;
    }
```

（12）编辑 CustomerOperation 类，输入如下代码，实现根据 typeId 查询等级信息的操作。

```
    /**
     * 查询等级信息
     *
     * @param typeId
     * @return
     */
    public String queryType(String typeId) {
        try {
            // 创建 Statement 对象，从 Type 表中查询等级属性
            java.sql.PreparedStatement prepar1 = connection
                    .prepareStatement("select Display from Type where TypeId=?");
            // 设置 Statement 对象的相应参数
            prepar1.setInt(1, Integer.parseInt(typeId));
            // 执行查询
            ResultSet res1 = prepar1.executeQuery();
            res1.next();
            return res1.getString("Display");
```

```
        } catch (SQLException ex) {

            return null;
        }
    }
```

（13）编辑 CustomerOperation 类，输入如下代码，将新增的订单信息插入到数据库中。

```
    /**
     * 插入订单信息
     *
     * @param ofb
     * @return
     */
    public boolean insertOrder(OrderFormBean ofb) {
        int flag = 0;
        try {
            // 创建 Statement 对象，向 OrderForm 表中插入一条新记录
            preparedStatement = connection
                    .prepareStatement("insert into OrderForm(UserId,BookId,amount,
OrderData,money) values(?,?,?,?,?)");
            // 设置其相应的参数
            preparedStatement.setInt(1, ofb.getUserId());
            preparedStatement.setInt(2, ofb.getBookId());
            preparedStatement.setInt(3, ofb.getAmount());
            preparedStatement.setString(4, ofb.getOrderData());
            preparedStatement.setInt(5, ofb.getMoney());
            // 执行更新
            flag = preparedStatement.executeUpdate();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        if (flag > 0) {
            return true;
        } else {
            return false;
        }
    }
```

（14）编辑 CustomerOperation 类，输入如下代码，关闭与数据库的连接。

```
    /**
     * 关闭与数据库的连接
     */
    public void Close() {
        try {
            if (resultSet != null) {
                // 关闭 ResultSet
                resultSet.close();
            }
            if (preparedStatement != null) {
```

```
                    // 关闭 Statement 对象
                    preparedStatement.close();
                }
                if (connection != null) {
                    // 关闭 Connection 对象
                    connection.close();
                }
            } catch (SQLException ex) {
                ex.printStackTrace();
            }

        }

}
```

CustomerOperation.java 类封装了网上书店中对数据库的所有操作，包括查询用户信息、书籍信息、等级信息等相应操作。

13.5　使用 JSP 实现后台管理系统的视图层

网上书店管理应用系统用 Strtus 框架实现了视图层和逻辑控制层，其视图部分以 JSP 作为实现手段，接受用户的输入并将结果反馈给用户。

在 shop 工程下创建“pages”文件夹，该文件夹用来存放系统中所有的 JSP 文件。在 pages 文件夹下创建名称为“admin”和“bookstore”两个文件夹。

13.5.1　创建用户登录页面

用户登录页面与用户交互，通过 username 和 password 文本框收集用户名和用户密码信息，并将这些信息提交给对应的 Action，处理用户的身份验证。

其中的<FORM action=adminLoginAction.do method=post>语句表明与 login.jsp 页面相对应的 adminLoginAction。

跟我做

（1）在“shop”工程的“admin”文件夹下创建“login.jsp”文件。

（2）编辑 login.jsp 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<title>网上书店管理系统</title>
```

```
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">
</head>
<body>
     <!--提交用户登录信息的 Form,提交的信息将由 adminLoginAction 来处理-->
<FORM action=adminLoginAction.do method=post >
<table width="400" border="0" align="center" cellpadding="0" cellspacing="1" bgcolor="#6685C5">

  <tr>
    <td bgcolor="#FFFFFF">
     <table width="400" border="0" align="center" cellpadding="0" cellspacing="0">
       <tr>
         <td colspan="3">   </td>
       </tr>
       <tr>
         <td height="35" colspan="3"><div align="center">用户名:
           <!--输入“用户名”文本框-->
         <INPUT   name=username   size=16 maxLength=16>
         用户密码:
         <!--输入“用户密码”文本框-->
         <INPUT name=password type=password size=16 maxLength=20>
         </div></td>
       </tr>
       <tr>
         <td width="100"></td>
         <td width="140">
           <!--“登录”按钮-->
           <input name=submit type=submit    value="登 录">
           <!--“取消”按钮-->
            <input name=submit1 type=reset    id="submit1" value="取 消"></td>
       </tr>

    </table>
    </td>
  </tr>
</table>
</FORM>
</body>
</html>
```

13.5.2　创建图书列表页面

bookList 页面将从数据库中查询的图书信息以列表的形式显示出来，<c:forEach>标签的作用就是迭代输出标签内部的内容。它既可以进行固定次数的迭代输出，也可以依据集合中对象的个数来决定迭代的次数。<c:forEach>标签的语法定义如下所示：

```
<c:forEach var="name" items="expression" varStatus="name"

          begin="expression" end="expression" step="expression">
```

```
            body content
</c:forEach>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“bookList.jsp”文件。

（2）编辑该文件，输入如下代码信息：

```
<%@page contentType="text/html;charset=gb2312"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!--加载自己定义的标记库-->
<%@taglib uri="/WEB-INF/mytags/MyTage.tld" prefix="my"%>
<jsp:include flush="false" page="checkAdmin.jsp"/>
<html>
<head>
<title>书籍列表</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">

<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">
</head>
<body text="#000000" topmargin=0>
<!--全部图书的列表-->
<table width="98%" border="0" cellpadding="2" cellspacing="0" align="center" class=TableBorder>
  <tr height="22" valign="middle" align="center">
    <th height="25" colspan="10">图书列表</th>
  </tr>
  <!--图书列表的表头-->
  <tr>
    <td width="4%" height="25" class=forumrow>
      <div align="center">ID</div>
    </td>
    <td width="15%" class=forumrow>
      <div align="center">书名</div>
    </td>
    <td width="6%" class=forumrow>
      <div align="center">出版社</div>
    </td>
    <td width="8%" class=forumrow>
      <div align="center">所属类型</div>
    </td>
    <td width="8%" class=forumrow>
      <div align="center">等级</div>
    </td>
    <td width="9%" class=forumrow>
      <div align="center">数量</div>
    </td>
```

```
    <td width="7%" class=forumrow>
      <div align="center">单价</div>
    </td>
    <td width="16%" class=forumrow>
      <div align="center">作者</div>
    </td>

  </tr>
  <!--循环显示所有的图书信息-->
  <c:forEach var="list" items="${requestScope.list}">
    <tr>
      <td height="25">
        <div align="center">${list.bookId}</div>
      </td>
      <td height="25">
        <div align="center">${list.bookName}</div>
      </td>
      <td>
        <div align="center">${list.bookCompany}            </div>
      </td>
      <td>
        <div align="center">${list.bookType}    </div>
      </td>
      <td>
           <!--使用自己定制的标记显示图书的等级-->
        <div align="center">    <my:bookStep>${list.bookStep}</my:bookStep>          </div>
      </td>
      <td>
        <div align="center">${list.bookStorage} </div>
      </td>
      <td>
        <div align="center">${list.bookPrice}元</div>
      </td>
      <td>
        <div align="center">${list.bookPenster}</div>
      </td>

    </tr>
  </c:forEach>
  <tr>
       <!--页面导航-->
    <td height="25" colspan="10">
      <div align="center">            当前页
${requestScope.page}            /总页数
${requestScope.pagecount}            <a href="BookPage.do?page=1" class="red">首页</a>
          <a href="BookPage.do?page=${requestScope.page-1<=0?"1":requestScope.page-1}"
class="red">上一页</a>
          <a href="BookPage.do?page=${requestScope.page+1>=requestScope.pagecount?request
Scope. pagecount:requestScope.page+1}" class="red">下一页</a>
```

```
            <a href="BookPage.do?page=${requestScope.pagecount}" class="red">末页</a>
          </div>
        </td>
      </tr>
</table>
</body>
</html>
```

13.5.3　创建添加图书信息页面

bookAdd.jsp 页面实现了新添图书信息的功能。本小节将创建这个页面。

该 JSP 页面实现了添加图书信息的功能，form1 的表单负责收集新图书信息，将这些信息提交后交由 InsertBook Action 来处理。

该页面使用数据库 JSTL 标记完成数据库查询操作。JSP 页面使用下面的指令导入这个库。

```
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
```

对于没有默认数据库的 JSP 页面，<sql:setDataSource>能够准备一个数据库以供使用。

```
<sql:setDataSource
driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"
user="sa"
password=""
var="db"
scope="request"/>
```

<sql:setDataSource>标签有如表 13-18 所示的属性。

表 13-18　setDataSource标签的属性

属　　性	说　　明	是 否 需 要	默　　认
driver	需要注册的 JDBC 驱动程序类的名称	不	无
url	用户数据库连接的 JDBC URL	不	无
user	数据库用户名	不	无
password	数据库密码	不	无
dataSource	预先准备的数据库	不	无
var	代表数据库的变量名	不	设置为默认
scope	代表数据库的变量的作用域	不	页面

JSTL 使用<sql:query>从数据库读取数据。核心标记<c:forEach>用于遍历结果集。<c:forEach>标记读取查询中的每一行。使用列名来获取行中的每一列的值。

```
<sql:query var="bookStep" dataSource="${requestScope.db}" sql="select * from BookStep"/>
```

该语句获取数据的等级信息。

跟我做

（1）在“shop”工程的“admin”文件夹下创建“bookAdd.jsp”文件。

（2）编辑 bookAdd.jsp 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312"%>
<!--加载 tag 库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<jsp:include flush="false" page="checkAdmin.jsp"/>
    <!--数据库连接信息-->
<sql:setDataSource                    driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"  user="sa"  password=""
var="db" scope="request"/>
<html>
<head>
<title>新增商品维护</title>
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">

<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
${requestScope.message}
</head>
<body bgcolor="#eff3f7" topmargin="0"  leftmargin="0">
    <!--收集新图书信息的 Form，提交后交由 InsertBookAction 来处理-->
<form name="form1" method="post" action=InsertBook.do>
<input type="hidden" name="par">
<br>
<table width="100%" border="0" cellspacing="0" cellpadding="0" class=TableBorder>
<tr> <td align="left" valign="middle" height="2"></td></tr>

<tr>
<td align="left" valign="top">
<table align=center border=1 bordercolor=#095ab5
                 bordercolordark=#ffffff cellpadding=0 cellspacing=0
                 id=submenu1 width="90%">
               <tr valign=center BGCOLOR="#D0ECF9">
                 <th align=CENTER height=25 valign=MIDDLE COLSPAN="4">
                     <font size="2"><b>图书增加</b></font></th>
               </tr>
               <tbody>
                 <tr valign=center>
                   <td align=left height=25 valign=MIDDLE
                       width="20%" colspan=1><font color="#000000">   
 商品名称：</font></td>
                   <td align=left height=25 valign=MIDDLE
                       colspan=3><font color="#000000">  
                           <!--输入书籍名称的文本框-->
                           <input type="text" name="BookName" />
                           </font></td>
```

```
                    </tr>

                    <tr valign=center>
                      <td align=left height=25 valign=MIDDLE
                          width="20%" colspan=1><font color="#000000">   
 商品类型：</font></td>
                      <td align=left height=25 valign=MIDDLE><font color="#000000"> 
                          <select name="BookType">
                              <!--从数据库中加载子类信息-->
                          <option value="all">请选择子类</option>
                           <sql:query var="bookType" dataSource="${requestScope.db}" sql="
select * from Type"/>
                                          <c:forEach var="f" items="${bookType.rows}">
                  <option value="${f.TypeId}">${f.Display}                </option>
                </c:forEach>
                          </select>
                          </font></td>
                      <td align=left height=25 valign=MIDDLE
                          width="20%" colspan=1><font color="#000000">   
 出版社：</font></td>
                      <td align=left height=25 valign=MIDDLE><font color="#000000"> 
                         <!--输入出版社信息的文本框-->
                           <input type="text" name="BookCompany" />
                          </font>
                          </td>
                    </tr>
                    <tr>
                      <td align=left height=25 valign=MIDDLE
                          width="20%" colspan=1><font color="#000000">   
 价   格：</font></td>
                      <td align=left height=25 valign=MIDDLE><font color="#000000"> 
                         <!--输入图书价格的文本框-->
                           <input type="text" name="BookPrice" />
                          </font>
                          </td>
                      <td align=left height=25 valign=MIDDLE
                          width="20%" colspan=1>   库存数量</td>
                      <td align=left height=25 valign=MIDDLE>  
                         <!--输入库存数量的文本框-->
                         <input type="text" name="BookStorage" />   </td>
                    </tr>
                    <tr valign=center>
                      <td align=left height=25 valign=MIDDLE
                          width="20%" colspan=1><font color="#000000">   
 作   
                         者：</font></td>
                      <td align=left height=25 valign=MIDDLE><font color="#000000"> 
                         <!--输入作者姓名的文本框-->
                           <input type="text" name="BookPenster" />
```

```
                                    </font>
                                    </td>
                              <td align=left height=25 valign=MIDDLE
                                    width="20%" colspan=1>    等级</td>
                              <td align=left height=25 valign=MIDDLE><font color="#000000"> 
                                   <!--从数据库中加载等级的相关信息-->
                              <sql:query var="bookStep" dataSource="${requestScope.db}" sql="select *
from BookStep"/>
                                    <select name="BookStep">
                                    <option value="all">请选择子类</option>
                                                                              <c:forEach var="s" items=
"${bookStep.rows}">
                        <option value="${s.BookStepId}">${s.Display}                    </option>
                     </c:forEach>
                                    </select></font>   </td>
                           </tr>
                           <tr valign=center>
                              <td align=left height=25 valign=MIDDLE
                                    width="20%" colspan=1><font color="#000000">   
 图   
                                   片          :</font><input          type=hidden          name=filePath
value="/shop/img/bookImage/emptybook.gif"></td>
                              <td align=left height=25 valign=MIDDLE
                                    colspan=3>
                                    <!--文件上载功能-->
                                   <iframe src=upload.jsp frameborder=0 marginheight=0 marginwidth=0
scrolling=no width="564" height="51" ></iframe></td>
                           </tr>
                           <tr valign=center>
                              <td align=left height=25 valign=MIDDLE
                                    width="20%" colspan=1><font color="#000000">   
 详细介绍：</font></td>
                              <td align=left height=25 valign=MIDDLE
                                    colspan=3><font color="#000000"> 
                                          <!--输入书籍的详细介绍-->
                                       <textarea name="BookSynopsis" cols="60" rows="15"></textarea>
                                    </font>
                              </td>
                           </tr>
                        </tbody>
                     </table>
                     <br>
                     <input type="submit" value="新增"></p></td></tr>
</table></td></tr> <tr> <td align="left" valign="top" background="images/line_hor.jpg" height="1">
</td></tr> </table>
<script>
function setPicPath(url)
{
      document.form1.filePath=url;
```

```
}
</script>
</form>
</body>
</html>
```

13.5.4　创建新增图书类型页面

通过 bookType.jsp 页面新增图书类型。本小节将创建这个页面。

该页面通过如下语句设置数据源：

```
<sql:setDataSource                    driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"   user="sa"   password=""
var="db" scope="request"/>
```

如下语句获取图书类型信息：

```
<sql:query var="bookType" dataSource="${requestScope.db}" sql="select * from Type"/>
```

跟我做

（1）在“shop”工程的“admi”文件夹下创建“bookType.jsp”文件。

（2）编辑 bookType 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312" %>
<!--加载 tag 库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<sql:setDataSource                    driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"   user="sa"   password=""
var="db" scope="request"/>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<meta http-equiv="refresh" content="60*30">
<title>网上书店管理系统</title>
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">
<script language="JavaScript" type="text/JavaScript">
function delpay()
{
    if(confirm("确定要删除此吗？"))
       return true;
    else
       return false;
}
</script>
</head>
```

```
<body topmargin=0>
<TABLE width=90% border=0 align=center cellPadding=5 cellSpacing=0 class="tableBorder">
    <TBODY>
    <TR>
        <Th colspan=3 align=center>图书类型</Th>
    </TR>
    <TR align=center bgcolor="#f1f3f5">
        <TD width=14%>编号</TD>
        <TD width=32%>费用类型名称</TD>
        <TD width=22%>操作</TD>
    </TR>
    <!--加载所有的类型信息-->
    <sql:query var="bookType" dataSource="${requestScope.db}" sql="select * from Type"/>
     <c:forEach var="f" items="${bookType.rows}">

    <TR>
        <TD><div align="center">${f.TypeId}</div></TD>
        <TD><div align="center">${f.Display}</div></TD>
        <!--对图书类型的操作-->
        <TD align=center><A HREF="bookType_edit.jsp?bookTypeId=${f.TypeId}">编辑</A>
        | <A HREF="DeleteBookType.do?bookTypeId=${f.TypeId}" onClick="return delpay();">
删除</A></TD>
    </TR>
            </c:forEach>
      <TR>
      <TD colspan="3"><div align="center">   </div></TD>
      </TR>
    </TBODY>
</TABLE>
<p>   </p>
<TABLE width=50% border="0" align=center cellpadding=4 cellspacing=1 class="tableBorder">
    <!--收集新的类型信息，提交后由 InserBookType Action 来处理-->
<form method="POST" action="InsertBookType.do" onsubmit="return check_input(this)">
    <TR>
        <Th colspan=2><div align="center">添加费用类型</div></th>
    </TR>
    <TR bgcolor="#FFFFFF">
      <TD width="30%">类型名称</TD>
        <TD width="70%">
      <input TYPE="text" id="Display" name="display" size=20 maxlength=20> **不得超过
10 个汉字</TD>
    </TR>

    <TR bgcolor="#FFFFFF"><TD height="45" colspan=2 align=center>    <FONT
color=#000000>
        <INPUT name=Submit type=submit value="确  定">   
        <INPUT name=Submit2 type=reset value="清  除"></FONT></TD>
    </TR>
```

```
  </form>
</TABLE>
</body>
</html>
```

该页面收集新的类型信息提交后由 InsertBookType Action 来处理，从而完成添加新的图书类型的操作。

13.5.5 创建显示图书分类信息页面

通过 bookTypeEdit 页面将显示所有的图书分类信息。本小节将创建该页面。

该页面通过如下语句设置数据源：

```
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password=""
var="db" scope="request"/>
```

如下语句获取商品类型信息：

```
<sql:query var="bookType" dataSource="${requestScope.db}" sql="select * from Type where
TypeId=${param.bookTypeId}"/>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“bookTypeEdit.jsp”文件。

（2）编辑 bookTypeEdit.jsp 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312" %>
<!--加载 tag 库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
 <!--数据库的连接信息-->
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password=""
var="db" scope="request"/>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<META HTTP-EQUIV="Expires" CONTENT="0">
<title>网上书店管理系统</title>
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">
</head>
<body topmargin=0>
<TABLE width=50% border="0" align=center cellpadding=4 cellspacing=1 class="tableBorder">
<form method="POST" action="UpdateBookType.do">
      <TR height=25>
```

```
            <Th colspan=2><div align="center" class="whitetitle"><B>编辑类型</B></div></Th>
      </TR>
      <TR bgcolor="#FFFFFF"><TD width="30%"> 类型名称</TD>
            <TD width="70%">
                  <!--从数据库中查询图书类型信息-->
            <sql:query var="bookType" dataSource="${requestScope.db}" sql="select * from Type
where TypeId=${param.bookTypeId}"/>
      <c:forEach var="f" items="${bookType.rows}">
         <input name=display TYPE="text" id="name" value="${f.Display}" size=20 maxlength=20>
         <input name=bookTypeId TYPE=hidden id="name" value="${param.bookTypeId}" size=20
maxlength=20>
         </c:forEach>
          **不得超过 10 个汉字</TD>
      </TR>
      <TR bgcolor="#FFFFFF"><TD colspan=2 align=center><BR>
         <FONT color=#000000>
            <INPUT name=Submit type=submit value="确  定">   
            <INPUT name=Submit2 type=reset value="清  除"></FONT><BR></TD>
      </TR>
   </form>
</TABLE>
```

该页面使用数据库 JSTL 标记完成数据库查询操作。将所有的图书类型信息以列表的形式列出。

13.5.6　创建订单列表页面

OrderList.jsp 页面以列表的形式显示所有的订单信息。本小节将创建这个页面。

<c:forEach>标签的作用就是迭代输出标签内部的内容。它既可以进行固定次数的迭代输出，也可以依据集合中对象的个数来决定迭代的次数。该页面中的如下代码将所有的订单信息列出：

```
<c:forEach var="list" items="${requestScope.list}">
   …
</c:forEach>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“OrderList.jsp”文件。

（2）编辑 OrderList.jsp 文件，输入如下代码信息：

```
<%@page contentType="text/html;charset=gb2312"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!--加载自己定义的标记库-->
<%@taglib uri="/WEB-INF/mytags/MyTage.tld" prefix="my"%>
<jsp:include flush="false" page="checkAdmin.jsp"/>
<html>
<head>
```

```
<title>网上书店管理系统</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">

<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">

</head>
<body text="#000000" topmargin=0>
     <!--全部订单的列表-->
<table width="98%" border="0" cellpadding="2" cellspacing="0" align="center" class=TableBorder>
   <tr height="22" valign="middle" align="center">
      <th height="25" colspan="10">订单</th>
   </tr>
   <!--订单列表的表头-->
   <tr>
      <td width="4%" height="25" class=forumrow>
         <div align="center">ID</div>
      </td>
      <td width="15%" class=forumrow>
         <div align="center">书名</div>
      </td>
      <td width="6%" class=forumrow>
         <div align="center">出版社</div>
      </td>
      <td width="8%" class=forumrow>
         <div align="center">所属类型</div>
      </td>
      <td width="9%" class=forumrow>
         <div align="center">数量</div>
      </td>
      <td width="7%" class=forumrow>
         <div align="center">总价</div>
      </td>
      <td width="16%" class=forumrow>
         <div align="center">用户名称</div>
      </td>
      <td width="17%" class=forumrow>
         <div align="center">用户城市</div>
      </td>
      <td width="9%" class=forumrow>
         <div align="center">日期</div>
      </td>
      <td width="9%" class=forumrow>
         <div align="center">操作</div>
      </td>
   </tr>
   <!--循环显示所有的订单信息-->
   <c:forEach var="list" items="${requestScope.list}">
```

```
  <tr>
    <td height="25">
      <div align="center">${list.bookId}</div>
    </td>
    <td height="25">
      <div align="center"><my:bookName>${list.bookId}</my:bookName></div>
    </td>
    <td>
      <div align="center"><my:bookcom>${list.bookId}</my:bookcom>          </div>
    </td>
    <td>
      <div align="center"><my:bookinfo>${list.bookId}</my:bookinfo></div>
    </td>
    <td>
      <div align="center">${list.amount}        </div>
    </td>
    <td>
      <div align="center">${list.money} </div>
    </td>
    <td>
     <!--使用自己定制的标记显示用户的名字-->
      <div align="center"><my:username>${list.userId}</my:username></div>
    </td>
    <td>
      <div align="center"><my:usercity>${list.userId}</my:usercity></div>
    </td>
    <td>
      <div align="center"><my:date>${list.orderData}</my:date></div>
    </td>
    <td>
      <div align="center">
        <a href="">已付款</a>
        <a href="pay_del.jsp?id=" onClick="return delpay();">删除</a>
      </div>
    </td>
  </tr>
    </c:forEach>
  <tr>
     <!--页面导航-->
    <td height="25" colspan="10">
      <div align="center">              当前页
${requestScope.page}              /总页数
${requestScope.pagecount}             <a href="OrderPage.do?page=1" class="red">首页</a>
        <a      href="OrderPage.do?page=${requestScope.page-1<=0?"1":requestScope.page-1}"
class="red">上一页</a>
        <a
href="OrderPage.do?page=${requestScope.page+1>=requestScope.pagecount?requestScope.
pagecount:requestScope.page+1}" class="red">下一页</a>
        <a href="OrderPage.do?page=${requestScope.pagecount}" class="red">末页</a>
```

```
        </div>
      </td>
    </tr>
</table>
</body>
</html>
```

该页面使用数据库 JSTL 标记完成数据库查询操作。将所有的图书订单信息以列表的形式列出。

13.5.7　创建用户列表页面

UserList.jsp 页面将所有的用户信息以列表的形式显示出来。<c:forEach>标签的作用就是迭代输出标签内部的内容。它既可以进行固定次数的迭代输出，也可以依据集合中对象的个数来决定迭代的次数。该页面中的如下代码将所有的用户信息列出。

```
<c:forEach var="list" items="${requestScope.list}">
  …
</c:forEach>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“UserList.jsp”文件。

（2）编辑 UserList.jsp 文件，输入如下代码信息：

```
<%@page contentType="text/html;charset=gb2312"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!--加载自己定义的标记库-->
<%@taglib uri="/WEB-INF/mytags/MyTage.tld" prefix="my"%>
<jsp:include flush="false" page="checkAdmin.jsp"/>
<html>
<head>
<title>网上书店管理系统</title>

<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
<META HTTP-EQUIV="Pragma" CONTENT="no-cache">
<META HTTP-EQUIV="Cache-Control" CONTENT="no-cache">
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">
</head>
<body text="#000000" topmargin=0>
            <!--全部用户的列表-->
<table width="98%" border="0" cellpadding="2" cellspacing="0" align="center" class=TableBorder>
  <tr height="22" valign="middle" align="center">
    <th height="25" colspan="10">用户列表</th>
  </tr>
   <!--用户列表的表头-->
  <tr>
    <td width="4%" height="25" class=forumrow>
      <div align="center">ID</div>
```

```
    </td>
    <td width="15%" class=forumrow>
      <div align="center">用户名</div>
    </td>
        <td width="15%" class=forumrow>
      <div align="center">真名</div>
    </td>
    <td width="6%" class=forumrow>
      <div align="center">城市</div>
    </td>
    <td width="8%" class=forumrow>
      <div align="center">用户等级</div>
    </td>
    <td width="8%" class=forumrow>
      <div align="center">订单数</div>
    </td>
    <td width="9%" class=forumrow>
      <div align="center">操作</div>
    </td>
  </tr>
  <!--循环显示所有的用户信息-->
  <c:forEach var="list" items="${requestScope.list}">
  <tr>
    <td height="25">
      <div align="center">${list.userId}</div>
    </td>
   <td height="25">
      <div align="center">${list.userName}</div>
    </td>
    <td height="25">
      <div align="center">${list.realName}</div>
    </td>
    <td>
     <!--使用自己定制的标记显示用户的城市名称-->
      <div align="center"><my:usercity>${list.userId}</my:usercity>        </div>
    </td>
    <td>
      <div align="center"><my:userstep>${list.userId}</my:userstep></div>
    </td>
    <td>
      <div align="center"><my:userordercount>${list.userId}</my:userordercount>        </div>
    </td>
    <td>
      <div align="center">
        <a href="user_edit.jsp?userId=${list.userId}">编辑</a>
        |
        <a href="DeleteUser.do?userId=${list.userId}" onClick="return delpay();">删除</a>
      </div>
    </td>
```

```
  </tr>
    </c:forEach>
  <tr>
     <!--页面导航-->
    <td height="25" colspan="10">
      <div align="center">           当前页
${requestScope.page}          /总页数
${requestScope.pagecount}           <a href="UserPage.do?page=1" class="red">首页</a>
        <a     href="UserPage.do?page=${requestScope.page-1<=0?"1":requestScope.page-1}"
class="red">上一页</a>
        <a
href="UserPage.do?page=${requestScope.page+1>=requestScope.pagecount?requestScope.
pagecount:requestScope.page+1}" class="red">下一页</a>
        <a href="UserPage.do?page=${requestScope.pagecount}" class="red">末页</a>
      </div>
    </td>
  </tr>
</table>
</body>
</html>
```

该页面使用数据库 JSTL 标记完成数据库查询操作，将所有的用户信息以列表的形式列出。

13.5.8　创建编辑用户信息页面

userEdit.jsp 页面用于编辑用户信息。该页面通过如下语句设置数据源：

```
<sql:setDataSource                        driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"   user="sa"   password=""
var="db" scope="request"/>
```

如下语句获取用户信息：

```
<sql:query var="users" dataSource="${db}">select * from users where userId=${param.userId}
</sql:query>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“userEdit.jsp”文件。

（2）编辑 userEdit.jsp 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312"%>
<!--加载 tag 库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<!--数据源的配置-->
```

```
<sql:setDataSource                         driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"   user="sa"   password=""
var="db" scope="request"/>
<!--查询相应的信息-->
<sql:query var="users" dataSource="${db}">select * from users where userId=${param.userId}
</sql:query>
<c:forEach var="u" items="${users.rows}">
<html:html>
<head>

<title>网上书店管理系统</title>
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#eff3f7" topmargin="0"   leftmargin="0">
<!--编辑用户信息的 Form，提交后由 EditUser 来处理-->
<html:form action="/admin/EditUser.do" method="post">
<html:hidden property="userId" value="${param.userId}"/>

<table   align=center   border=1   bordercolor=#095ab5   bordercolordark=#ffffff   cellpadding=0
cellspacing=0 id=submenu1 width="34%" class="tableBorder">
                <tr valign=center BGCOLOR="#D0ECF9">
                  <th align=CENTER height=25 valign=MIDDLE>
                       <font size="2">用户编辑</font></th>
                </tr>
                <tbody>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 用    户    名 :
</font><html:text readonly="true" property="userName" value="${u.userName}"/></td>
                  </tr>

                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 密              码 :
</font><html:password property="userPwd" value="${u.UserPwd}" size="20"/></td>
                  </tr>
                  <tr>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 真  实  姓  名 :
</font><html:text property="realName" value="${u.realName}" size="20"/></td>
                  </tr>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 年              龄 :
</font><html:text property="userAge" value="${u.UserAge}" size="20"/></td>
                  </tr>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
```

```
                        width="61%" colspan=1><font color="#000000">性      别：</font>男
<html:radio value="1" property="userSex"/>女<html:radio value="0" property="userSex"/></td>
                  </tr>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 电 话 号 码 ：
</font><html:text property="userPhone"   value="${u.UserPhone}"   size="20"/></td>
                  </tr>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 身 份 证 号 ：
</font><html:text property="userCard" value="${u.Usercard}"   size="20"/></td>
                  </tr>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 邮 箱 地 址 ：
</font><html:text property="userMail" value="${u.UserMail}"   size="20"/></td>
                  </tr>
                  <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 来 自 何 方 ：
</font><html:select   property="userCity"     value="${u.UserCity}">   <sql:query   var="query"
dataSource="${db}">select * from City</sql:query>
                <c:forEach var="i" items="${query.rows}">
                  <html:option value="${i.CityId}">${i.Display}</html:option></c:forEach>
</html:select></td>
                  </tr>
                   <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 具 体 地 址 ：
</font><html:text property="userAdds" value="${u.UserAdds}" size="20"/></td>
                  </tr>

                       <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 邮 政 编 码 ：
</font><html:text property="userCode" value="${u.UserCode}" size="20"/></td>
                  </tr>

              <tr valign=center>
                    <td align=left height=25 valign=MIDDLE
                        width="61%"   colspan=1><font   color="#000000"> 所 在 公 司 ：
</font><html:text property="userWork" value="${u.UserWork}" size="20"/></td>
                  </tr>

               </tbody>
              </table>

              <html:submit value="更新" property=""/></p>
```

```
</html:form>

</body>
</html:html></c:forEach>
```

13.5.9　创建添加管理员页面

adminAdd.jsp 页面用于添加新的管理员信息。该页面通过如下语句设置数据源：

```
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password=""
var="db" scope="request"/>
```

如下语句获取管理员信息：

```
<sql:query var="admin" dataSource="${requestScope.db}" sql="select * from admin"/>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“adminAdd.jsp”文件。

（2）编辑 adminAdd.jsp 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312"%>
<!--加载 tag 库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
    <!--数据库连接信息-->
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password=""
var="db" scope="request"/>
<html>
<head>
<title>网上书店系统</title>
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">

<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#eff3f7" topmargin="0"  leftmargin="0">
<table align=center width="299" class="tableBorder">
<TBODY>
    <TR>
        <Th colspan=2 align=center>管理员</Th>
    </TR>
    <TR align=center bgcolor="#f1f3f5">
        <TD width="57%">管理员编号</TD>
        <TD width="37%">管理员账号</TD>
    </TR>
    <!--列出所有的管理员用户信息-->
```

```
        <sql:query var="admin" dataSource="${requestScope.db}" sql="select * from admin"/>
        <c:forEach var="a" items="${admin.rows}">

        <TR>
                <TD><div align="center">${a.userId}</div></TD>
                <TD><div align="center">${a.userName}</div></TD>
        </TR>
       </c:forEach>
        </TBODY>
</table>
<!--收集管理员信息，提交后交由 InsertAdmin 来处理-->
<form name="editUserForm" method="post" action="InsertAdmin.do">
<table  align=center  border=1  bordercolor=#095ab5  bordercolordark=#ffffff  cellpadding=0
cellspacing=0 id=submenu1 width="34%" class="tableBorder">
                            <tr valign=center BGCOLOR="#D0ECF9">
                              <th align=CENTER height=25 valign=MIDDLE>
                                        <font size="2">添加管理员</font></th>
                            </tr>
                            <tbody>
                              <tr valign=center>
                                <td align=left height=25 valign=MIDDLE
                                        width="61%"  colspan=1><font  color="#000000"> 用  户  名 :
</font><input type="text" name="userName" value=""></td>
                              </tr>

                              <tr valign=center>
                                <td align=left height=25 valign=MIDDLE
                                        width="61%" colspan=1><font color="#000000">密   
码：</font><input type="password" name="userPwd" size="20" value=""></td>



                              </tr>
                              <tr>
                                <td align=left height=25 valign=MIDDLE
                                        width="61%"  colspan=1> 确 认 <font  color="#000000"> 密 码 :
</font><input type="password" name="userPwd" size="20" value=""></td>
                              </tr>

                            </tbody>
                          </table>
                          <input type="submit" name="" value="增加"></p>
</form>
</body>
</html>
```

该页面添加了新的管理员信息。

13.5.10　创建修改管理员信息页面

changeAdmin.jsp 页面用于修改管理员信息。本小节将创建该页面。

该页面通过如下语句设置数据源：

```
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver" url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password="" var="db" scope="request"/>
```

如下语句获取管理员信息：

```
<sql:query var="admin" dataSource="${requestScope.db}" sql="select * from admin"/>
```

跟我做

（1）在“shop”工程的“admin”文件夹下创建“changeAdmin.jsp”文件。

（2）编辑 changeAdmin.jsp 文件，输入如下代码信息：

```
<%@ page contentType="text/html;charset=gb2312"%>
<!--加载 tag 库-->
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
    <!--数据库连接信息-->
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver" url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password="" var="db" scope="request"/>
<html>
<head>
<title>网上书店系统</title>
<link rel="stylesheet" href="images/css.css" type="text/css" media="screen">

<meta http-equiv="Content-Type" content="text/html; charset=gb2312">
</head>
<body bgcolor="#eff3f7" topmargin="0"  leftmargin="0">
<table align=center width="299" class="tableBorder">
<TBODY>
    <TR>
        <Th colspan=2 align=center>管理员</Th>
    </TR>
    <TR align=center bgcolor="#f1f3f5">
        <TD width="57%">管理员编号</TD>
        <TD width="37%">管理员账号</TD>
    </TR>
    <!--列出所有的管理员用户信息-->
    <sql:query var="admin" dataSource="${requestScope.db}" sql="select * from admin"/>
    <c:forEach var="a" items="${admin.rows}">
```

```
    <TR>
        <TD><div align="center">${a.userId}</div></TD>
        <TD><div align="center">${a.userName}</div></TD>
    </TR>
   </c:forEach>
    </TBODY>
</table>
<!--收集管理员信息，提交后交由 InsertAdmin 来处理-->
<form name="editUserForm" method="post" action="InsertAdmin.do">
<table  align=center  border=1  bordercolor=#095ab5  bordercolordark=#ffffff  cellpadding=0
cellspacing=0 id=submenu1 width="34%" class="tableBorder">
              <tr valign=center BGCOLOR="#D0ECF9">
                <th align=CENTER height=25 valign=MIDDLE>
                     <font size="2">添加管理员</font></th>
              </tr>
              <tbody>
                <tr valign=center>
                  <td align=left height=25 valign=MIDDLE
                      width="61%"  colspan=1><font  color="#000000"> 用  户  名 ：
</font><input type="text" name="userName" value=""></td>
                </tr>

                <tr valign=center>
                  <td align=left height=25 valign=MIDDLE
                      width="61%" colspan=1><font color="#000000">密   
码：</font><input type="password" name="userPwd" size="20" value=""></td>


                </tr>
                <tr>
                  <td align=left height=25 valign=MIDDLE
                      width="61%"  colspan=1> 确 认 <font  color="#000000"> 密 码 ：
</font><input type="password" name="userPwd" size="20" value=""></td>
                </tr>

             </tbody>
            </table>
  <input type="submit" name="" value="增加"></p>
</form>
</body>
</html>
```

该页面完成管理员的修改密码的操作。

13.6　自定义标签的实现

JSP 标签是用来调用 JavaBean 组件的操作，处理定向请求以简化 JSP 页面开发与维护。

JSP 技术提供了一种封装其他动态类型的机制——自定义标签，它扩展了 JSP 语言。自定义标签通常发布在标签库中，该库定义了一个自定义标签集并包含实现标签的对象。

自定义标签是用户定义的 JSP 语言元素。当 JSP 页面包含一个自定义标签时被转换为 Servlet，标签转换为称为 tag handler 的对象的操作。接着当 Servlet 执行时 Web container 调用那些操作。自定义标签有着丰富的特点，它们可以：

- 通过调用页面传递的属性进行自定义。
- 访问对于 JSP 页面可能的所有对象。
- 修改由调用页面产生的响应。
- 相互间通信。可以创建并初始化一个 JavaBean 组件，创建一个变量引用标签中的 bean，接着在其他的标签中引用该 bean。
- 在一个标签中嵌套另一个，可以在 JSP 页面中进行复杂的交互。

要定义一个标签，需要为标签开发一个标签处理累计帮助类以及在标签库中声明标签描述符。如果标签处理器需要与体交互，标签处理器必须实现 BodyTag，这样的处理器实现了方法 doInitBody，doAfterBody。这些方法与传递到标签处理器的体内容交互。

跟我做

（1）在“shop”工程中创建“tag”包，该包将包含所有有关自定义标签的 Java 类。

（2）在“tag”包中创建 DateOptionTag.java 类，编辑该文件，输入如下代码信息：

```
public class DateOptionTag   extends BodyTagSupport
{
    JspWriter out=null;

    /**
     * 覆盖 doAfterBody()类
     * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
     */
    public int doAfterBody()
    {
     //取得 BodyContent 对象
        BodyContent body=getBodyContent();
        //取得 Body 中的内容
        String content=body.getString();
        //取得 JspWriter 对象
        out=body.getEnclosingWriter();
        try {
          //取得正确的日期格式
            out.print(content.substring(0,4)+"年"+content.substring(4,6)+"月"+content.substring
(6,8)+"号");

        } catch (IOException e) {
            e.printStackTrace();
        }

        return (SKIP_BODY);
```

```
        }
}
```

该类继承了 BodyTagSupport 类，取得 body 的内容后转换成特定的日期格式。

（3）在“tag”包中创建 GetBookCom.java 类，编辑该文件，输入如下代码信息：

```
public class GetBookCom extends BodyTagSupport {
        JspWriter out = null;

        /**
         * 覆盖 doAfterBody()类
         *
         * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
         */
        public int doAfterBody() {
                // 取得 BodyContent 对象
                BodyContent body = getBodyContent();
                // 取得 Body 中的内容
                String content = body.getString();
                // 取得 JspWriter 对象
                out = body.getEnclosingWriter();
                // 创建 AdminOperation 对象
                AdminOperation db = new AdminOperation();
                int bookId = Integer.parseInt(content);
                // 从数据库中查询特定图书出版社信息
                String name = db.getBookBean(bookId).getBookCompany();
                // 关闭与数据库的操作
                db.Close();
                try {
                        out.print(name);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return (SKIP_BODY);
        }
}
```

该类继承了 BodyTagSupport 类，body 中的内容是 Bookid，根据该信息从数据库中查询到对应的出版社信息。

（4）在“tag”包中创建 GetBookInfo.java 类，编辑该文件，输入如下代码信息：

```
public class GetBookInfo extends BodyTagSupport {
        JspWriter out = null;

        /**
         * 覆盖 doAfterBody()类
         *
         * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
         */
        public int doAfterBody() {
```

```
            // 取得 BodyContent 对象
            BodyContent body = getBodyContent();
            // 取得 Body 中的内容
            String content = body.getString();
            // 取得 JspWriter 对象
            out = body.getEnclosingWriter();
            // 初始化 AdminOperation 对象
            AdminOperation db = new AdminOperation();
            int bookId = Integer.parseInt(content);
            // 根据 bookId 从数据库中查询图书类型的信息
            String name = db.getBookBean(bookId).getBookType();
            // 关闭数据库连接
            db.Close();
            try {
                out.print(name);
            } catch (IOException e) {
                e.printStackTrace();
            }
            return (SKIP_BODY);
        }
}
```

该类继承了 BodyTagSupport 类，body 中的内容是 Bookid，根据该信息从数据库中查询到对应的图书类型信息。

（5）在“tag”包中创建 GetBookName.java 类，编辑该文件，输入如下代码信息：

```
public class GetBookName extends BodyTagSupport {
        JspWriter out = null;

        /**
         * 覆盖 doAfterBody()类
         *
         * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
         */
        public int doAfterBody() {
            // 取得 BodyContent 对象
            BodyContent body = getBodyContent();
            // 取得 Body 中的内容
            String content = body.getString();
            // 取得 JspWriter 对象
            out = body.getEnclosingWriter();
            // 初始化 AdminOperation 对象
            AdminOperation db = new AdminOperation();
            int bookId = Integer.parseInt(content);
            // 根据 bookId 从数据库中查询图书书名的信息
            String name = db.getBookBean(bookId).getBookName();
            // 关闭数据库连接
            db.Close();
            try {
                out.print(name);
```

```
            } catch (IOException e) {
                e.printStackTrace();
            }
            return (SKIP_BODY);
        }
}
```

该类继承了 BodyTagSupport 类，body 中的内容是 Bookid，根据该信息从数据库中查询到对应的图书名称信息。

（6）在“tag”包中创建 GetBookStep.java 类，编辑该文件，输入如下代码信息：

```
public class GetBookStep extends BodyTagSupport {
        JspWriter out = null;

        /**
         * 覆盖 doAfterBody()类
         *
         * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
         */
        public int doAfterBody() {
                // 取得 BodyContent 对象
                BodyContent body = getBodyContent();
                // 取得 Body 中的内容
                String content = body.getString();
                // 取得 JspWriter 对象
                out = body.getEnclosingWriter();
                // 初始化 AdminOperation 对象
                AdminOperation db = new AdminOperation();
                int setpId = Integer.parseInt(content);
                // 根据 stepId 从数据库中查询图书等级的信息
                String name = db.getBookStep(setpId);
                // 关闭数据库连接
                db.Close();
                try {
                        out.print(name);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return (SKIP_BODY);
        }
}
```

该类继承了 BodyTagSupport 类，body 中的内容是 stepid，根据该信息从数据库中查询到对应的图书等级信息。

（7）在“tag”包中创建 GetUserCity.java 类，编辑该文件，输入如下代码信息：

```
public class GetUserCity extends BodyTagSupport {
        JspWriter out = null;

        /**
```

```
         * 覆盖 doAfterBody()类
         *
         * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
         */
        public int doAfterBody() {
                // 取得 BodyContent 对象
                BodyContent body = getBodyContent();
                // 取得 Body 中的内容
                String content = body.getString();
                // 取得 JspWriter 对象
                out = body.getEnclosingWriter();
                // 初始化 CustomerOperation 对象
                CustomerOperation db = new CustomerOperation();
                int userId = Integer.parseInt(content);
                // 根据 userId 从数据库中查询用户城市的信息
                int cityId = db.getUser(userId).getUserCity();
                String name = db.getCity(cityId);
                // 关闭数据库的连接
                db.Close();
                try {
                        out.print(name);
                } catch (IOException e) {
                        e.printStackTrace();
                }
                return (SKIP_BODY);
        }
}
```

该类继承了 BodyTagSupport 类，body 中的内容是 userid，根据该信息从数据库中查询到对应的用户城市信息。

（8）在“tag”包中创建 GetUserName.java 类，编辑该文件，输入如下代码信息：

```
public class GetUserName extends BodyTagSupport {
        JspWriter out = null;

        /**
         * 覆盖 doAfterBody()类
         *
         * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
         */
        public int doAfterBody() {
                // 取得 BodyContent 对象
                BodyContent body = getBodyContent();
                // 取得 Body 中的内容
                String content = body.getString();
                // 取得 JspWriter 对象
                out = body.getEnclosingWriter();
                // 初始化 CustomerOperation 对象
                CustomerOperation db = new CustomerOperation();
                int userId = Integer.parseInt(content);
```

```
        // 根据 userId 从数据库中查询用户的真实姓名
        String name = db.getUser(userId).getRealName();
        // 关闭数据库的连接
        db.Close();
        try {
            out.print(name);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return (SKIP_BODY);
    }
}
```

该类继承了 BodyTagSupport 类，body 中的内容是 userid，根据该信息从数据库中查询到对应的用户的真实姓名信息。

（9）在“tag”包中创建 GetUserOrderCount.java 类，编辑该文件，输入如下代码信息：

```
public class GetUserOrderCount extends BodyTagSupport {
    JspWriter out = null;

    /**
     * 覆盖 doAfterBody()类
     *
     * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
     */
    public int doAfterBody() {
        // 取得 BodyContent 对象
        BodyContent body = getBodyContent();
        // 取得 Body 中的内容
        String content = body.getString();
        // 取得 JspWriter 对象
        out = body.getEnclosingWriter();
        // 初始化 CustomerOperation 对象
        CustomerOperation db = new CustomerOperation();
        int userId = Integer.parseInt(content);
        // 根据 userId 从数据库中查询该用户订单的数量
        int resuilt = db.getUserOrderCount(userId);
        // 关闭数据库的连接
        db.Close();
        try {
            out.print(resuilt);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return (SKIP_BODY);
    }

}
```

该类继承了 BodyTagSupport 类，body 中的内容是 userid，根据该信息从数据库中查询到对应的用户的订单数量。

（10）在“tag”包中创建 GetUserStep.java 类，编辑该文件，输入如下代码信息：

```
public class GetUserStep extends BodyTagSupport {
	JspWriter out = null;

	/**
	 * 覆盖 doAfterBody()类
	 *
	 * @see javax.servlet.jsp.tagext.IterationTag#doAfterBody()
	 */
	public int doAfterBody() {
		// 取得 BodyContent 对象
		BodyContent body = getBodyContent();
		// 取得 Body 中的内容
		String content = body.getString();
		// 取得 JspWriter 对象
		out = body.getEnclosingWriter();
		CustomerOperation db = new CustomerOperation();
		// 初始化 CustomerOperation 对象
		int userId = Integer.parseInt(content);
		// 根据 userId 从数据库中查询该用户的等级
		String name = db.getStep(db.getUser(userId).getUserStrp());
		// 关闭数据库的连接
		db.Close();
		try {
			out.print(name);
		} catch (IOException e) {
			e.printStackTrace();
		}
		return (SKIP_BODY);
	}

}
```

该类继承了 BodyTagSupport 类，body 中的内容是 userid，根据该信息从数据库中查询到对应的用户的等级信息。

（11）在“shop”工程中创建“MyTag.tld”文件，编辑该文件，输入如下代码信息：

```
<?xml version="1.0" encoding="UTF-8"?>
<taglib xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/
web-jsptaglibrary_2_0.xsd" version="2.0">
	<description>BookStep tag</description>
	<display-name>Name</display-name>
	<tlib-version>1.0</tlib-version>
	<short-name>BName</short-name>
	<tag>
```

```
        <description>通过 ID 获取图书的等级</description>
        <name>bookStep</name>
        <tag-class>tags.GetBookStep</tag-class>
        <body-content>JSP</body-content>
 </tag>
<tag>
        <description>通过 ID 获取图书的书名</description>
        <name>bookName</name>
        <tag-class>tags.GetBookName</tag-class>
        <body-content>JSP</body-content>
 </tag>
<tag>
        <description>通过 ID 获取图书出版社</description>
        <name>bookcom</name>
        <tag-class>tags.GetBookCom</tag-class>
        <body-content>JSP</body-content>
 </tag>
<tag>
        <description>通过 ID 获取图书信息</description>
        <name>bookinfo</name>
        <tag-class>tags.GetBookInfo</tag-class>
        <body-content>JSP</body-content>
 </tag>
      <tag>
        <description>通过 ID 获取用户姓名</description>
        <name>username</name>
        <tag-class>tags.GetUserName</tag-class>
        <body-content>JSP</body-content>
 </tag>
<tag>
        <description>通过 ID 获取用户城市信息</description>
        <name>usercity</name>
        <tag-class>tags.GetUserCity</tag-class>
        <body-content>JSP</body-content>
 </tag>
      <tag>
        <description>通过 ID 获取日期</description>
        <name>date</name>
        <tag-class>tags.DateOptionTag</tag-class>
        <body-content>JSP</body-content>
 </tag>
            <tag>
        <description>通过 ID 获取用户等级</description>
        <name>userstep</name>
        <tag-class>tags.GetUserStep</tag-class>
        <body-content>JSP</body-content>
 </tag>                  <tag>
        <description>通过 ID 获取用户的订单数量</description>
        <name>userordercount</name>
```

```
            <tag-class>tags.GetUserOrderCount</tag-class>
            <body-content>JSP</body-content>
      </tag>
</taglib>
```

标签库描述符是 XML 格式的文档。TLD 包含库的所有信息及库中的每个标签。TLD 文件必须以扩展名.tld 为后缀。

13.7　创建后台管理系统的 ActionForm

ActionForm 代表了由浏览器所传递过来的数据。每个 ActionForm 中定义的属性应该与页面中表单的控件或者说页面中所提交的数据相对应。这样，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。所有的 ActionForm 类都继承于 org.apache.struts.validator.ValidatorForm 类。

13.7.1　创建编辑用户信息的 ActionForm

EditUserForm 为典型的 JavaBean，包括用户真实姓名、用户地址等属性。对于每个属性都包括 getter/setter 方法。该 ActionForm 用于收集用户的信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。

跟我做

在“shop”工程的“src”文件夹中创建“actionform”包，在该包中创建 EditUserForm.java 类，编辑该文件，输入如下代码信息：

```
/**
 * 编辑用户信息的 ActionForm
 *
 */
public class EditUserForm extends ActionForm {
      // userId 属性
      private int userId;

      // realName 属性
      private String realName;

      // userAdds 属性
      private String userAdds;

      // userAge 属性
      private String userAge;
```

```
// userCard 属性
private String userCard;

// userCity 属性
private String userCity;

// userCode 属性
private String userCode;

// userMail 属性
private String userMail;

// userName 属性
private String userName;

// userPhone 属性
private String userPhone;

// userPwd 属性
private String userPwd;

// userSex 属性
private String userSex;

// userWork 属性
private String userWork;

/**
 * userWork 属性的 setter 方法
 *
 * @param userWork
 */
public void setUserWork(String userWork) {
	this.userWork = userWork;
}

/**
 * userSex 属性的 setter 方法
 *
 * @param userSex
 */
public void setUserSex(String userSex) {
	this.userSex = userSex;
}

/**
 * userPwd 的 setter 方法
 * @param userPwd
 */
```

```
public void setUserPwd(String userPwd) {
        this.userPwd = userPwd;
}

/**
 * userPhone 的 setter 方法
 * @param userPhone
 */
public void setUserPhone(String userPhone) {
        this.userPhone = userPhone;
}

/**
 * userName 的 setter 方法
 * @param userName
 */
public void setUserName(String userName) {
        this.userName = userName;
}

/**
 * userMail 的 setter 方法
 * @param userMail
 */
public void setUserMail(String userMail) {
        this.userMail = userMail;
}

/**
 * userCode 的 setter 方法
 * @param userCode
 */
public void setUserCode(String userCode) {
        this.userCode = userCode;
}

/**
 * userCity 的 setter 方法
 * @param userCity
 */
public void setUserCity(String userCity) {
        this.userCity = userCity;
}

/**
 * userCard 的 setter 方法
 * @param userCard
 */
```

```
    public void setUserCard(String userCard) {
        this.userCard = userCard;
    }

    /**
     * userAge 的 setter 方法
     * @param userAge
     */
    public void setUserAge(String userAge) {
        this.userAge = userAge;
    }

    /**
     * userAdds 的 setter 方法
     * @param userAdds
     */
    public void setUserAdds(String userAdds) {
        this.userAdds = userAdds;
    }

    /**
     * realName 的 setter 方法
     * @param realName
     */
    public void setRealName(String realName) {
        this.realName = realName;
    }

    /**
     * userId 的 setter 方法
     * @param userId
     */
    public void setUserId(int userId) {

        this.userId = userId;
    }

    /**
     * userId 的 getter 方法
     * @return
     */
    public int getUserId() {

        return userId;
    }

    /**
     * realName 的 getter 方法
     * @return
```

```
     */
    public String getRealName() {
          return realName;
    }

    /**
     * userAdds 的 getter 方法
     * @return
     */
    public String getUserAdds() {
          return userAdds;
    }

    /**
     * userAge 的 getter 方法
     * @return
     */
    public String getUserAge() {
          return userAge;
    }

    /**
     * userCard 的 getter 方法
     * @return
     */
    public String getUserCard() {
          return userCard;
    }

    /**
     * userCity 的 getter 方法
     * @return
     */
    public String getUserCity() {
          return userCity;
    }

    /**
     * userCode 的 getter 方法
     * @return
     */
    public String getUserCode() {
          return userCode;
    }

    /**
     * userMail 的 getter 方法
     * @return
     */
```

```
public String getUserMail() {
        return userMail;
}

/**
 * userName 的 getter 方法
 * @return
 */
public String getUserName() {
        return userName;
}

/**
 * userPhone 的 getter 方法
 * @return
 */
public String getUserPhone() {
        return userPhone;
}

/**
 * userPwd 的 getter 方法
 * @return
 */
public String getUserPwd() {
        return userPwd;
}

/**
 * userSex 的 getter 方法
 * @return
 */
public String getUserSex() {
        return userSex;
}

/**
 * userWork 的 getter 方法
 * @return
 */
public String getUserWork() {
        return userWork;
}

public ActionErrors validate(ActionMapping actionMapping,
                HttpServletRequest httpServletRequest) {
        /** @todo: finish this method, this is just the skeleton. */
        return null;
}
```

```
    public void reset(ActionMapping actionMapping,
                HttpServletRequest servletRequest) {
    }
}
```

13.7.2　创建收集图书信息的 ActionForm

BookInfActionForm 为典型的 JavaBean，包括书名、出版社等属性。对于每个属性都包括 getter/setter 方法。该 ActionForm 用于收集新增图书的信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。

跟我做

（1）在“shop”工程的“actionform”文件夹下创建“BookinfActionForm.java”文件。

（2）编辑 BookinfActionForm.java 类，输入如下代码信息：

```
public class BookInfActionForm extends ActionForm {
    // bookId 属性
    private String bookId;

    // bookCompany 属性
    private String bookCompany;

    // bookName 属性
    private String bookName;

    // bookPenster 属性
    private String bookPenster;

    // bookPrice 属性
    private String bookPrice;

    // bookStorage 属性
    private String bookStorage;

    // bookSynopsis 属性
    private String bookSynopsis;

    // bookType 属性
    private String bookType;

    /**
     * bookCompany 的 getter 方法
     *
     * @return
     */
    public String getBookCompany() {
```

```
		return bookCompany;
	}

	/**
	 * bookCompany 的 setter 方法
	 *
	 * @param bookCompany
	 */
	public void setBookCompany(String bookCompany) {
		this.bookCompany = bookCompany;
	}

	/**
	 * bookSynopsis 属性的 setter 方法
	 *
	 * @param bookSynopsis
	 */
	public void setBookSynopsis(String bookSynopsis) {
		this.bookSynopsis = bookSynopsis;
	}

	/**
	 * bookStorage 属性的 setter 方法
	 *
	 * @param bookStorage
	 */
	public void setBookStorage(String bookStorage) {
		this.bookStorage = bookStorage;
	}

	/**
	 * bookPrice 的 setter 方法
	 *
	 * @param bookPrice
	 */
	public void setBookPrice(String bookPrice) {
		this.bookPrice = bookPrice;
	}

	/**
	 * bookPenster 的 setter 方法
	 *
	 * @param bookPenster
	 */
	public void setBookPenster(String bookPenster) {
		this.bookPenster = bookPenster;
	}

	/**
```

```
     * bookName 的 setter 方法
     *
     * @param bookName
     */
    public void setBookName(String bookName) {
        this.bookName = bookName;
    }

    /**
     * bookId 的 setter 方法
     *
     * @param bookId
     */
    public void setBookId(String bookId) {
        this.bookId = bookId;
    }

    /**
     * bookType 的 setter 方法
     *
     * @param bookType
     */
    public void setBookType(String bookType) {
        this.bookType = bookType;
    }

    /**
     * bookName 的 getter 方法
     *
     * @return
     */
    public String getBookName() {
        return bookName;
    }

    /**
     * bookPenster 的 getter 方法
     *
     * @return
     */
    public String getBookPenster() {
        return bookPenster;
    }

    /**
     * bookPrice 的 getter 方法
     *
     * @return
     */
```

```
public String getBookPrice() {
        return bookPrice;
}

/**
 * bookStorage 的 getter 方法
 *
 * @return
 */
public String getBookStorage() {
        return bookStorage;
}

/**
 * bookSynopsis 的 getter 方法
 *
 * @return
 */
public String getBookSynopsis() {
        return bookSynopsis;
}

/**
 * bookId 的 getter 方法
 *
 * @return
 */
public String getBookId() {
        return bookId;
}

/**
 * bookType 的 getter 方法
 *
 * @return
 */
public String getBookType() {
        return bookType;
}

public ActionErrors validate(ActionMapping actionMapping,
                HttpServletRequest httpServletRequest) {
        /** @todo: finish this method, this is just the skeleton. */
        return null;
}

public void reset(ActionMapping actionMapping,
                HttpServletRequest servletRequest) {
```

```
        }
}
```

13.7.3　创建用户登录 ActionForm

LoginActionForm 为典型的 JavaBean，包括用户名、用户密码等属性。对于每个属性都包括 getter/setter 方法。该 ActionForm 用于收集用户的登录信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。

跟我做

（1）在“shop”工程的“actionfor”文件夹下创建“LoginActionForm.java”文件。

（2）编辑 LoginActionForm.java 文件，输入如下代码信息：

```
public class LoginActionForm extends ActionForm {
        // userName 属性
        private String userName;

        // userPwd 属性
        private String userPwd;

        /**
         * userName 属性的 getter 方法
         *
         * @return
         */
        public String getUserNmae() {
                return userNmae;
        }

        /**
         * userName 属性的 setter 方法
         *
         * @param userNmae
         */
        public void setUserNmae(String userNmae) {
                this.userNmae = userNmae;
        }

        /**
         * userPwd 的 setter 方法
         *
         * @param userPwd
         */
        public void setUserPwd(String userPwd) {
                this.userPwd = userPwd;
        }
```

```
	/**
	 * userPwd 的 getter 方法
	 *
	 * @return
	 */
	public String getUserPwd() {
		return userPwd;
	}

	public ActionErrors validate(ActionMapping actionMapping,
			HttpServletRequest httpServletRequest) {
		/** @todo: finish this method, this is just the skeleton. */
		return null;
	}

	public void reset(ActionMapping actionMapping,
			HttpServletRequest servletRequest) {
	}
}
```

13.8　实现后台管理系统的控制层

Action 的作用是接受用户的请求，通过调用业务方法实现业务处理的功能。在编写 Action 时必须要继承自 org.apache.struts.action.Action 类，并覆盖父类中的 execute()方法。这里的 execute()方法就是响应用户请求的处理方法，它是被 Struts 框架自动调用的。当用户在页面中提交表单后，Struts 框架就会把用户请求转发给 Action 组件。

本节实现网上书店后台管理系统中的所有 Action 类，实现其控制层。

跟我做

（1）在“shop”工程中的“src”文件夹创建“action”包，用于存放系统中所有的 Action 类。

（2）在“action”包中创建 AdminLoginAction.java 类，编辑该文件，输入如下代码信息：

```
public class AdminLoginAction extends Action {
	/**
	 * 覆盖 Action 的 execute()方法
	 *
	 */
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response) {
		// 创建 AdminBean 对象
		AdminBean ab = new AdminBean();
		// 创建 AdminOperation 对象
```

```
            AdminOperation db = new AdminOperation();
            // 设置 AdminBean 对象的 userName 属性
            ab.setUserName(request.getParameter("username"));
            // 设置 AdminBean 对象的 userName 属性
            ab.setUserPwd(request.getParameter("password"));
            // 通过 AdminOperation 对象验证该用户的合法性
            if (db.checkAdminLogin(ab)) {
                  // 将 AdminBean 对象存放到 request 对象中
                  request.getSession().setAttribute("admin", ab);
            }
            // 关闭数据库的连接
            db.Close();
            return mapping.findForward("adminIndex");
      }
}
```

该 Action 从 request 中取得 username 和 password，封装 AdminBean 对象后，通过 AdminOperation 对象验证该用户的合法性。

（3）在“action”包中创建 AdminLoginOut.java 类，编辑该文件，输入如下代码信息：

```
public class AdminLoginOut extends Action {
      /**
       * 覆盖 Action 的 execute()方法
       *
       */
      public ActionForward execute(ActionMapping mapping, ActionForm form,
                  HttpServletRequest request, HttpServletResponse response) {
            // 从 Session 中删除 admin 属性
            request.getSession().removeAttribute("admin");
            return mapping.findForward("adminIndex");
      }
}
```

该 Action 实现了管理员登出操作。从 Session 中去掉 admin 属性的值。

（4）在“action”包中创建 BookPageAction.java 类，编辑该文件，输入如下代码信息：

```
public class BookPageAction extends Action {
      /**
       * 覆盖 Action 的 execute()方法
       *
       */
      public ActionForward execute(ActionMapping mapping, ActionForm form,
                  HttpServletRequest request, HttpServletResponse response) {
            int count = 10;
            int page = 1;
            //创建 AdminOperation 对象
            AdminOperation db = new AdminOperation();
            if (request.getParameter("page") != null) {
                  page = Integer.parseInt(request.getParameter("page"));
                  page = page == 0 ? 1 : page;
```

```
			}
			//通过 AdminOperation 对象从数据库查询到所有的图书信息，并保存到链表中
			ArrayList list = db.getBooksList(count, page);
			//将 page 信息保存到 request 中
			request.setAttribute("page", page + "");
			//将 pagecount 信息保存到 request 中
			request.setAttribute("pagecount", db.getSearchCountPage() + "");
			//将图书列表保存到 request 中
			request.setAttribute("list", list);
			//关闭数据库的连接
			db.Close();
			return mapping.findForward("booklist");
	}
}
```

该 Action 通过 AdminOperation 对象从数据库中查询所有的图书信息，并将必要的信息保存到 request 对象中。

（5）在“action”包中创建“ChangeAdminPwd.java”文件，编辑该文件，输入如下代码信息：

```
public class ChangeAdminPwd extends Action {
	/**
	 * 覆盖 Action 的 execute()方法
	 *
	 */
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response) {
		// 从 request 中取得 admin
		AdminBean admin = (AdminBean) request.getSession()
				.getAttribute("admin");
		// 从 request 中取得 olduserPwd
		String oldpwd = request.getParameter("olduserPwd");
		// 从 request 中取得 newuserPwd
		String newPwd = request.getParameter("newuserPwd");
		// 从 request 中取得 isuserPwd
		String isPwd = request.getParameter("isuserPwd");
		// 判断原始密码不正确
		if (!oldpwd.equals(admin.getUserPwd())) {
			request.setAttribute("message",
					"<script>alert('原始密码不正确  修改失败')</script>");
			return mapping.findForward("chengeadmin");
		} else if (!newPwd.equals(isPwd)) {
			// 判断新密码和确认密码是否一致
			request.setAttribute("message",
					"<script>alert('新密码和确认密码不一致  修改失败')</script>");
			return mapping.findForward("chengeadmin");
		}
		// 创建 AdminOperation 对象
		AdminOperation db = new AdminOperation();
```

```
            // 修改 admin 的密码为新密码
            admin.setUserPwd(newPwd);
            if (db.changeAdminPwd(admin)) {
                request.setAttribute("message", "<script>alert('更新成功');</script>");
                request.getSession().setAttribute("admin", admin);
            } else {
                request.setAttribute("message", "<script>alert('更新失败');</script>");
            }
            // 关闭数据库的连接
            db.Close();
            return mapping.findForward("chengeadmin");
        }
}
```

该 Action 完成了管理员密码的修改。

（6）在“action”包中创建 DeleteBookType.java 类，编辑该文件，输入如下代码信息：

```
public class DeleteBookType extends Action {
    /**
     * 覆盖 Action 的 execute()方法
     *
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 从 request 中取得 bookTypeId 信息
        int bookTypeId = Integer.parseInt(request.getParameter("bookTypeId"));
        // 创建 AdminOperation 对象
        AdminOperation db = new AdminOperation();
        // 通过 AdminOperation 从数据库中删除某个图书类型信息
        if (db.deleteBookType(bookTypeId)) {
            request.setAttribute("message", "<script>alert('删除成功');</script>");
        } else {
            request.setAttribute("message", "<script>alert('删除失败');</script>");
        }
        // 关闭数据库连接
        db.Close();
        return mapping.findForward("booktype");
    }
}
```

该 Action 通过 AdminOperation 对象从数据库中删除特定的图书类型信息。

（7）在“action”包中创建 EditUserAction.java，编辑该文件，输入如下代码信息：

```
public class EditUser extends Action {
    /**
     * 覆盖 Action 的 execute()方法
     *
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
```

```
		// 将 form 进行适当的类型转换
		EditUserForm editUserForm = (EditUserForm) form;
		// 创建 UserBean 对象
		UserBean ub = new UserBean();
		// 设置 UserBean 的相应属性的值
		ub.setRealName(editUserForm.getRealName());
		ub.setUserAdds(editUserForm.getUserAdds());
		ub.setUserAge(Integer.parseInt(editUserForm.getUserAge()));
		ub.setUserCard(editUserForm.getUserCard());
		ub.setUserCity(Integer.parseInt(editUserForm.getUserCity()));
		ub.setUserCode(Integer.parseInt(editUserForm.getUserCode()));
		ub.setUserId(editUserForm.getUserId());
		ub.setUserMail(editUserForm.getUserMail());
		ub.setUserName(editUserForm.getUserName());
		ub.setUserPhone(editUserForm.getUserPhone());
		ub.setUserPwd(editUserForm.getUserPwd());
		ub.setUserSex(Integer.parseInt(editUserForm.getUserSex()));
		ub.setUserWork(editUserForm.getUserWork());
		// 创建 CustomerOperation 对象
		CustomerOperation db = new CustomerOperation();
		// 更新 UserBean 的值到数据库中
		if (db.updateUser(ub)) {
			request.setAttribute("message", "<script>alert('更新成功');</script>");
		} else {
			request.setAttribute("message", "<script>alert('更新失败');</script>");
		}
		// 关闭数据库连接
		db.Close();
		return mapping.findForward("UserPage");
	}
}
```

该 Action 完成更新用户信息。其中的 UserBean 是典型的 JavaBean，包括用户的所有信息，其核心代码如下：

```
public class UserBean {
	// userName 属性
	private String userName;

	// userPwd 属性
	private String userPwd;

	// realName 属性
	private String realName;

	// userSex 属性
	private int userSex;

	// userPhone 属性
	private String userPhone;
```

```
    // userMail 属性
    private String userMail;

    // userCity 属性
    private int userCity;

    // userAdds 属性
    private String userAdds;

    // userCode 属性
    private int userCode;

    // userWork 属性
    private String userWork;

    // userCard 属性
    private String userCard;

    // userAge 属性
    private int userAge;

    // userStep 属性
    private int userStep;

    // userId 属性
    private int userId;
```

（8）在“action”包中创建 InsertAdmin.java 类，编辑该文件，输入如下代码信息：

```
public class InsertAdmin extends Action {
    /**
     * 覆盖 Action 的 execute()方法
     *
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 创建 AdminBean 对象
        AdminBean admin = new AdminBean();
        // 为 AdminBean 对象设置 UserName 属性
        admin.setUserName(request.getParameter("userName"));
        // 为 AdminBean 对象设置 UserPwd 属性
        admin.setUserPwd(request.getParameter("userPwd"));
        // 创建 AdminOperation 对象
        AdminOperation db = new AdminOperation();
        // 通过 AdminOperation 将 AdminBean 对象插入到数据库中
        if (db.insertAdmin(admin)) {
            request.setAttribute("message", "<script>alert('添加成功');</script>");
        } else {
            request.setAttribute("message", "<script>alert('添加失败');</script>");
```

```
            }
            // 关闭数据库连接
            db.Close();
            return mapping.findForward("addadmin");
        }
}
```

该 Action 完成将管理员信息插入到数据库中。其中的 AdminBean 是典型的 JavaBean，包括与管理员身份有关的信息，其核心代码如下：

```
public class AdminBean {
        // userId 属性
        private int userId;

        // userName 属性
        private String userName;

        // userPwd 属性
        private String userPwd;

        public AdminBean() {
        }

        /**
         * userId 的 setter 方法
         *
         * @param userId
         */
        public void setUserId(int userId) {
                this.userId = userId;
        }

        /**
         * userName 的 setter 方法
         *
         * @param userName
         */
        public void setUserName(String userName) {
                this.userName = userName;
        }

        /**
         * userPwd 的 setter 方法
         *
         * @param userPwd
         */
        public void setUserPwd(String userPwd) {
                this.userPwd = userPwd;
        }
```

```
	/**
	 * userId 的 getter 方法
	 *
	 * @return
	 */
	public int getUserId() {
		return userId;
	}

	/**
	 * userName 属性的 getter 方法
	 *
	 * @return
	 */
	public String getUserName() {
		return userName;
	}

	/**
	 * userPwd 属性的 getter 方法
	 *
	 * @return
	 */
	public String getUserPwd() {
		return userPwd;
	}
}
```

（9）在“action”包中创建 InsertBook.java 类，编辑该文件，输入如下代码信息：

```
public class InsertBook extends Action {
	/**
	 * 覆盖 Action 的 execute()方法
	 *
	 */
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response) {
		// 创建 BookBean 对象
		BookBean bookBean = new BookBean();
		// 设置 BookBean 对象的 bookName 属性
		bookBean.setBookName(request.getParameter("BookName"));
		// 设置 BookBean 对象的 bookType 属性
		bookBean.setBookType(request.getParameter("BookType"));
		// 设置 BookBean 对象的 bookCompany 属性
		bookBean.setBookCompany(request.getParameter("BookCompany"));
		bookBean.setBookPrice(Integer.parseInt(request
				.getParameter("BookPrice")));
		bookBean.setBookStorage(Integer.parseInt(request
				.getParameter("BookStorage")));
		bookBean.setBookPenster(request.getParameter("BookPenster"));
```

```
            bookBean
                    .setBookStep(Integer.parseInt(request.getParameter("BookStep")));
            bookBean.setBookImage(request.getParameter("filePath"));
            bookBean.setBookSynopsis(request.getParameter("BookSynopsis"));
            // 创建 AdminOperation 对象
            AdminOperation db = new AdminOperation();
            // 将 bookBean 对象的信息插入到数据库中
            if (db.insertBooks(bookBean)) {
                request.setAttribute("message", "<script>alert('添加成功');</script>");
            } else {
                request.setAttribute("message", "<script>alert('添加失败');</script>");
            }
            return mapping.findForward("bookadd");
        }
}
```

该 Action 从 request 中得到相关信息封装成 BookBean 对象，然后通过 AdminOperation 对象的 insertBooks 操作将这些信息插入到数据库中。

BookBean 是典型的 JavaBean 对象，包括与图书相关的属性，其核心代码如下：

```
public class BookBean {
    // bookId 属性
    private int bookId;

    // bookName 属性
    private String bookName;

    // bookPenster 属性
    private String bookPenster;

    // bookCompany 属性
    private String bookCompany;

    // bookSynopsis 属性
    private String bookSynopsis;

    // bookStorage 属性
    private int bookStorage;

    // bookSell 属性
    private int bookSell;

    // bookData 属性
    private Date bookData;

    // bookPrice 属性
    private int bookPrice;

    // bookType 属性
    private String bookType;
```

```
        // bookImage 属性
        private String bookImage;

        // bookStep 属性
        private int bookStep;

        // bookCount 属性
        private int bookCount;

        // bookAllPrice 属性
        private int bookAllPrice;

}
```

（10）在“action”包中创建 InsertBookType.java 类，编辑该文件，输入如下代码信息：

```
public class InsertBookType extends Action {
        /**
         * 覆盖 Action 的 execute()方法
         *
         */
        public ActionForward execute(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response) {
                // 创建 BookTypeBean 对象
                BookTypeBean btb = new BookTypeBean();
                // 设置 BookTypeBean 的属性
                btb.setDisplay(request.getParameter("display"));
                // 创建 AdminOperation 对象
                AdminOperation db = new AdminOperation();
                // 通过 AdminOperation 对象将 BookTypeBean 对象的信息插入到数据库中
                if (db.insertBookType(btb)) {
                        request.setAttribute("message", "<script>alert('添加成功');</script>");
                } else {
                        request.setAttribute("message", "<script>alert('添加失败');</script>");
                }
                // 关闭数据库的连接
                db.Close();
                return mapping.findForward("booktype");
        }
}
```

该 Action 从 request 中取得 display 的信息，封装成 BookTypeBean 对象，通过 AdminOperation 对象将这个对象的信息保存到数据库中。

其中的 BookTypeBean 是典型的 JavaBean，封装了 BookType 的 typeId 和 display 属性。其核心代码如下：

```
public class BookTypeBean {
        // typeId 属性
        private int typeId;
```

```
	// display 属性
	private String display;

	public BookTypeBean() {
	}

	/**
	 * typeId 属性的 setter 方法
	 *
	 * @param typeId
	 */
	public void setTypeId(int typeId) {
		this.typeId = typeId;
	}

	/**
	 * display 属性的 setter 方法
	 *
	 * @param display
	 */
	public void setDisplay(String display) {
		this.display = display;
	}

	/**
	 * typeId 属性的 getter 方法
	 *
	 * @return
	 */
	public int getTypeId() {
		return typeId;
	}

	/**
	 * display 属性的 getter 方法
	 *
	 * @return
	 */
	public String getDisplay() {
		return display;
	}
}
```

（11）在“action”包中创建“OrderPage.java”文件，编辑该文件，输入如下代码信息：

```
public class OrderPage extends Action {
	/**
	 * 覆盖 Action 的 execute()方法
	 *
```

```
	 */
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response) {
		int count = 10;
		int page = 1;
		// 创建 AdminOperation 对象
		AdminOperation db = new AdminOperation();
		if (request.getParameter("page") != null) {
			page = Integer.parseInt(request.getParameter("page"));
			page = page == 0 ? 1 : page;
		}
		// 通过 AdminOperation 从数据库中读取所有的订单信息
		ArrayList list = db.getOrderList(count, page);
		// 将 page、pagecount、list 等信息保存到 request 中
		request.setAttribute("page", page + "");
		request.setAttribute("pagecount", db.getSearchCountPage() + "");
		request.setAttribute("list", list);
		// 关闭数据库的连接
		db.Close();
		return mapping.findForward("orderlist");

	}
}
```

该 Action 通过 AdminOperation 从数据库中读取所有的订单信息，并将这些信息保存到 request 中。

（12）在“action”包中创建“UpdateBookType.java”文件，编辑该文件，输入如下代码信息：

```
public class UpdateBookType extends Action {
	/**
	 * 覆盖 Action 的 execute()方法
	 *
	 */
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response) {
		// 创建 BookTypeBean 对象
		BookTypeBean btb = new BookTypeBean();
		// 设置 BookTypeBean 对象的相关属性
		btb.setDisplay(request.getParameter("display"));
		btb.setTypeId(Integer.parseInt(request.getParameter("bookTypeId")));
		// 创建数据库操作的 AdminOperation 对象
		AdminOperation db = new AdminOperation();
		// 通过 AdminOperation 对象更新 BookType 信息
		if (db.updateBookType(btb)) {
			request.setAttribute("message", "<script>alert('更新成功');</script>");
		} else {
			request.setAttribute("message", "<script>alert('更新失败');</script>");
		}
```

```
            // 关闭数据库的连接
            db.Close();
            return mapping.findForward("booktype");

      }
}
```

该 Action 通过 AdminOperation 对象更新 BookType 信息。

（13）在“action”包中创建 UserPage.java 类，编辑该文件，输入如下代码信息：

```
public class UserPage extends Action {
      /**
       * 覆盖 Action 的 execute()方法
       *
       */
      public ActionForward execute(ActionMapping mapping, ActionForm form,
                  HttpServletRequest request, HttpServletResponse response) {
            int count = 10;
            int page = 1;
            // 创建 AdminOperation 对象
            AdminOperation db = new AdminOperation();
            if (request.getParameter("page") != null) {
                  page = Integer.parseInt(request.getParameter("page"));
                  page = page == 0 ? 1 : page;
            }
            // 通过 AdminOperation 对象取得所有的用户列表
            ArrayList list = db.getUsersList(count, page);
            // 将 page、pagecount、list 等信息保存到 request 中
            request.setAttribute("page", page + "");
            request.setAttribute("pagecount", db.getSearchCountPage() + "");
            request.setAttribute("list", list);
            // 关闭数据库的连接
            db.Close();
            return mapping.findForward("userlist");

      }
}
```

该 Action 通过 AdminOperation 对象取得所有的用户列表。

13.9　使用 JSP 实现前台展示系统的视图层

前台展示系统是网上书店的主要部分，网上书店系统与 Web 用户的所有交互都是通过前台展示部分实现的，包括实现 Web 用户的注册、显示所有的图书信息、显示特价图书信息以及创建购物车功能等。本节介绍如何实现网上书店前台展示系统的用户表现层。

13.9.1　创建用户注册页面

reg.jsp 页面为用户注册页面。该页面通过如下语句设置数据源：

```
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver" url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password="" var="db" scope="request"/>
```

如下语句获取所有的城市信息：

```
<sql:query var="query" dataSource="${db}">select * from City</sql:query>
```

跟我做

在“jsp”文件夹下的“bookstore”文件夹下创建“reg.jsp”文件，编辑该文件，输入如下代码信息：

```
  <c:if test="${requestScope.regok!=1}">
    <html:form action="/regAction.do"   method="post">
      <sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver" url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password="" var="db" scope="request"/>
      <div align="center">
        <table>
          <tr>
            <td>
              <font color="#996633">用 户 名：</font>
            </td>
            <td>
              <!--输入用户名信息-->
              <html:text property="userName"/>
            </td>
          </tr>
          <tr>
            <td>
              <font color="#996633">密码：
              </font>
            </td>
            <td>
              <!--输入用户密码信息-->
              <html:password property="userPwd"/>
            </td>
          </tr>
          <tr>
            <td>
              <font color="#996633">确认密码：</font>
            </td>
            <td>
              <!--输入确认密码信息-->
```

```
            <html:password property="userPwd1"/>
        </td>
    </tr>
    <tr>
        <td>
            <font color="#996633">真实姓名：</font>
        </td>
        <td>
            <!--输入用户真实姓名信息-->
            <html:text property="realName"/>
        </td>
    </tr>
    <tr>
        <td>
            <font color="#996633">年龄：
            </font>
        </td>
        <td>
            <!--输入用户年龄信息-->
            <html:text property="userAge"/>
        </td>
    </tr>
    <tr>
        <td>
            <font color="#996633">性别：
            </font>
        </td>
        <!--选择用户性别-->
        <td>                    男
            <html:radio value="1" property="userSex"/>
                                女
            <html:radio value="2" property="userSex"/>
        </td>
    </tr>
    <tr>
        <td>
            <font color="#996633">电话号码：</font>
        </td>
        <td>
            <!--输入用户电话号码-->
            <html:text property="userPhone"/>
        </td>
    </tr>
    <tr>
        <td>
            <font color="#996633">身份证号：</font>
        </td>
        <td>
            <!--输入用户身份证号-->
```

```
            <html:text property="userCard"/>
          </td>
        </tr>
        <tr>
          <td>
            <font color="#996633">邮箱地址：</font>
          </td>
          <td>
            <!--输入用户 E-mail 信息-->
            <html:text property="userMail"/>
          </td>
        </tr>
        <tr>
          <td>
            <font color="#996633">来自何方：</font>
          </td>
          <td>
            <!--从数据库中查询所有的城市信息-->
            <html:select property="userCity" title="请选择">
              <sql:query var="query" dataSource="${db}">select * from City</sql:query>
              <c:forEach var="i" items="${query.rows}">
                <html:option value="${i.CityId}">${i.Display}                    </html:option>
              </c:forEach>
            </html:select>
          </td>
        </tr>
        <tr>
          <td>
            <font color="#996633">具体地址：</font>
          </td>
          <td>
            <!--输入用户具体地址-->
            <html:text property="userAdds"/>
          </td>
        </tr>
        <tr>
          <td>
            <font color="#996633">邮政编码：</font>
          </td>
          <td>
            <!--输入用户邮政编码-->
            <html:text property="userCode"/>
          </td>
        </tr>
        <tr>
          <td>
            <font color="#996633">所在公司：</font>
          </td>
          <td>
```

```
                <!--输入用户所在公司信息-->
                <html:text property="userWork"/>
              </td>
            </tr>
            <tr align="center">
              <td align="center" colspan="2">
                <br/>
                <html:image property="" onclick="document.form1.submit()" src="/shop/img/xx.
gif"/>
              </td>
            </tr>
          </table>
        </div>
      </html:form>
    </c:if>
```

该页面完成用户的注册，其界面如图 13-15 所示，包括用户名、密码、年龄、电话号码等信息。

用 户 名：
密　　码：
确认密码：
真实姓名：
年　　龄：
性　　别：男 ○　　女 ○
电话号码：
身份证号：
邮箱地址：
来自何方：北京
具体地址：
邮政编码：
所在公司：
提交信息

图 13-15　用户登录界面

13.9.2　创建显示图书信息页面

bookinfo.jsp 页面将显示所有的图书信息。该页面通过如下语句设置数据源：

```
<sql:setDataSource driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop" user="sa" password=""
var="db" scope="request"/>
```

如下语句获取所有的图书信息：

```
<!--查询所有图书信息-->
      <sql:query var="query" dataSource="${requestScope.db}" sql="select * from information
where bookId=?">
```

```
        <!--查询语句的参数-->
          <sql:param value="${param.bookID}"/>
        </sql:query>
```

然后将查询到结果显示出来。

跟我做

在“bookstore”文件夹下创建“bookinf.jsp”文件，编辑该文件，其核心代码如下：

```
      <c:if test="${not empty param.bookID}">
      <!--显示所有的图书信息-->
        <form action="/bookInfAction.do" method="POST">
        <!--设置数据源-->
        <sql:setDataSource                    driver="com.microsoft.jdbc.sqlserver.SQLServerDriver"
url="jdbc:microsoft:sqlserver://127.0.0.1:1433;databasename=MyShop"   user="sa"   password=""
var="db" scope="request"/>
        <!--查询所有图书信息-->
        <sql:query  var="query"  dataSource="${requestScope.db}"  sql="select  *  from  information
where bookId=?">
        <!--查询语句的参数-->
          <sql:param value="${param.bookID}"/>
        </sql:query>
        <c:forEach var="row" items="${query.rows}">
          <td valign="top" bordercolor="#ffffff" width="410" bgcolor="#ffffff">
            <div align="center">
              <table cellspacing="0" cellpadding="4" width="100%" align="center" border="0">
                <tr>
                  <td align="middle" colspan="4">
                    <font color="#ff0000">
                      <b>图书信息</b>
                    </font>
                  </td>
                </tr>
                <tr>
                  <td background="/shop/img/bj_x1.gif" colspan="4" height="1">   </td>
                </tr>
              </table>
              <table cellspacing="0" cellpadding="0" width="100%" border="0">
                <tr>
                  <td>
                    <table cellspacing="0" cellpadding="0" width="100%" border="0">
                      <tr>
                        <td valign="top" width="160">
                          <table cellspacing="0" cellpadding="2" width="100%" border="0">
                            <tr>
                              <td>
                                <div align="center">
                                  <a href="${row.bookImage}" target="_blank">
                                    <img   height="130"   alt="  点  击  浏  览  图  书  大  图  "
src="${row.bookImage}" width="130" border="0">
```

```
                        </a>
                      </div>
                    </td>
                  </tr>
                  <tr>
                    <td>
                      <div align="center">
                        <a href="${row.bookImage}" target="_blank">
                          <img   height="36"   src="/shop/img/view2.gif"   width="115"
border="0">
                        </a>
                      </div>
                    </td>
                  </tr>
                  <tr>
                    <td align="middle">                              会员价：
${row.BookPrice}                              元
                    </td>
                  </tr>
                  <tr>
                    <td align="middle">                              VIP 价：
                      <font color="#ff9999">更优惠</font>
                    </td>
                  </tr>
                </table>
              </td>
              <td valign="top">
                <table cellspacing="0" cellpadding="2" width="100%" border="0">
                  <br>
                  <tr>
                    <td height="30">
                      <font color="#990000" size="2">
                        <strong>${row.BookName}              <br   />      <br   />
</strong>
                      </font>
                    </td>
                  </tr>
                  <tr>

                    <td>【图书作者】 ${row.bookPenster} </td>
                  </tr>
                  <tr>
                    <td>                              【出版社】
                     ${row.bookCompany}
                    </td>
                  </tr>
                  <tr>
                    <td>                              【库存】
${row.BookStorage}                              本
```

```
                                </td>
                            </tr>
                            <tr>
                                <td>                              【所属类别】
                                    <sql:query var="Type" dataSource="${requestScope.db}" sql=
"select * from Type where TypeId=${row.BookType}"/>
                                    <c:forEach var="e" items="${Type.rows}">
                                        <a href="/shop/pagerAction.do?TypeId=${e.TypeId} &bookStep
=0" targer="_blank">${e.Display}</a>
                                    </c:forEach>
                                </td>
                            </tr>
                             <tr>
                                <td>                              【图书等级】
                                    <sql:query  var="BookStep"  dataSource="${requestScope.db}"
sql="select * from BookStep where BookStepId=${row.BookStep}"/>
                                    <c:forEach var="step" items="${BookStep.rows}">
                                        <a href="/shop/pagerAction.do?TypeId=0&bookStep=${step.
BookStepId}" targer="_blank">${step.Display}                    </a>
                                    </c:forEach>
                                </td>
                            </tr>
                            <tr>
                                <td>       有 0 位网友评论</td>
                            </tr>
                        </table>
                        <table height="80">
                            <tr>
                                <td valign="bottom" align="middle" width="221">
                                    <a
onclick="javascript:window.open('/shop/bookInfAction.do?bookId=${row.BookId}','gouwu','width=
450,height=300,toolbar=no, status=no, menubar=no, resizable=yes, scrollbars=yes');" href=
"javascript:;">
                                        <img  height="36"  src="/shop/img/shopcart2.gif"  width="100"
border="0" alt="">
                                    </a>
                                </td>
                            </tr>
                        </table>
                    </td>
                </tr>
            </table>
```

该 Jsp 页面从数据库中查询到所有的图书信息，并分类显示出图书信息。

13.9.3　创建显示特价图书信息页面

tejia.jsp 页面显示所有的特价图书信息。该页面通过“select top 10 *from information where bookStep=3 order by BookId DESC”语句从数据库中查询到所有的特价图书信息。

```
<sql:query var="tejia" dataSource="${requestScope.db}" sql="select top 10 *from information
where bookStep=3 order by BookId DESC"/>
```

跟我做

在“bookstore”文件夹下创建 tejia.jsp 文件，编辑该文件，其核心代码如下：

```
<sql:query var="tejia" dataSource="${requestScope.db}" sql="select top 10 *from information
where bookStep=3 order by BookId DESC"/>
        <td><%-- 特价图书--%>
        <c:forEach var="tej" items="${tejia.rows}">
         <table cellspacing="0" cellpadding="0" width="100%" border="0">
            <tr>
            <td background="/shop/img/dp_bg.gif" height="24">
                  <a title="${tej.BookName}" href="/shop/jsp/bookinf.jsp?bookID=${tej.bookId}">
                                <c:if test="${fn:length(tej.BookName) > 10}">${fn:substring(tej. Book
Name,0,10)}
                                </c:if>
                                <c:if     test="${fn:length(tej.BookName)     <=     10}">${tej.BookName}
</c:if>
                        </a>
             </td>
            </tr>
          </table>
        </c:forEach>
```

该页面显示所有的特价图书信息。

13.9.4　创建购物车页面

购物车是网上书店应用系统的关键部件。该页面实现了一个购物车，用户将打算购买的图书放进购物车中，并显示图书总数和总价值。页面实现了 updateshoppingcart()和 detail()两个 JavaScript 方法，分别实现更新购物车内容和显示购物车购物细节的功能。

```
<form name="shopcar" method="post" action="/shop/jsp/orderInsertAction.do">
```

语句中指定了购物车的信息提交后由 orderInsertAction 来执行用户新购图书的插入功能。

跟我做

在“bookstore”文件夹下创建 shopcar.jsp 页面，编辑该文件，输入如下代码信息：

```
<%@page contentType="text/html; charset=GBK"%>
<%@taglib uri="/WEB-INF/struts-bean.tld" prefix="bean"%>
<%@taglib uri="/WEB-INF/struts-html.tld" prefix="html"%>
<%@taglib uri="/WEB-INF/struts-logic.tld" prefix="logic"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x"%>
<html>
<head>
<title>图书商城--我的购物车</title>
<meta http-equiv="Content-Type" content="text/html; charset=gb2312">

<link href="/shop/img/css.css" rel="stylesheet" type="text/css">

<script language="javascript" type="">
            function updateshoppingcart()
            {
                document.shopcar.action = "shopCarAction.do?action=countChenge";
                document.shopcar.submit();
            }
          function   delall()
          {
            document.shopcar.action = "shopCarAction.do?action=delall";
            document.shopcar.submit();
          }
      </script>
</head>
<body>
<table align="center" border="0" cellpadding="2" cellspacing="2" width="96%">
  <tr>
    <td align="center" valign="baseline">我的购物车</td>
  </tr>
</table>
<form name="shopcar" method="post" action="/shop/jsp/orderInsertAction.do">
<table align="center" bgcolor="#ffcc00" border="0" cellpadding="1" cellspacing="1" width="96%">

  <tr bgcolor="#ffe2a6">
    <td height="22" width="25%">
      <div align="center">
        <font color="#666666">商品名称</font>
      </div>
    </td>
    <td height="22" width="15%">
      <div align="center">
        <font color="#666666">单价 (会员)</font>
      </div>
    </td>
    <td bgcolor="#ffe2a6" height="22" width="15%">
```

```
        <div align="center">
          <font color="#666666">数量</font>
        </div>
      </td>
      <td height="22" width="15%">
        <div align="center">
          <font color="#666666">总价</font>
        </div>
      </td>
      <td height="22" width="10%">
        <div align="center">
          <font color="#666666">删除</font>
        </div>
      </td>
    </tr>
    <c:forEach var="i" items="${sessionScope.list}">
      <tr bgcolor="#fff9ec">
        <td style="padding-left: 5px;" height="22" width="25%">
          <div align="left">
            <a href="jsp/bookinf.jsp?bookID=${i.bookId}" target="_blank">${i.bookName}         </a>
            <input name="bookId" value="${i.bookId}" type="hidden">
          </div>
        </td>
        <td height="22" width="15%">
          <div align="center">${i.bookPrice}              </div>
        </td>
        <td height="22" width="15%">
          <div align="center">
            <font color="#dd6600">
              <input type="text" name="${i.bookId}" value="${i.bookCount}" size="6" class="form">
            </font>
          </div>
        </td>
        <td height="22" width="15%">
          <div align="center">
            <font color="#ff3300">${i.bookAllPrice}</font>
            元
          </div>
        </td>
        <td height="22" width="10%">
          <div align="center">
            <a href="/shop/shopCarAction.do?action=del&bookId=${i.bookId}">
              <img src="/shop/img/trash.gif" border="0" height="17" width="15" alt="">
            </a>
          </div>
        </td>
      </tr>
    </c:forEach>
    <tr bgcolor="#ffffff">
```

```
    <td colspan="6" height="25">
      <div align="center">
        <a onclick="javascript:window.close();">
          <img alt="" src="/shop/img/cart01.gif" border=0/>
        </a>
        <a onclick="updateshoppingcart()">
          <img alt="" src="/shop/img/cart03.gif" border=0/>
        </a>
        <a   onclick="delall()">
          <img alt="" src="/shop/img/cart02.gif" border=0/>
        </a>
        <a   onclick="javascript:document.shopcar.submit();">
          <img alt="" src="/shop/img/cart04.gif" border=0/>
        </a>
      </div>
    </td>
  </tr>
  <tr bgcolor="#fff9ec">
    <td colspan="6" height="36">
      <div  align="center">           购物车里有商品：${sessionScope.shangpin}件  总数：
${sessionScope.zongshu}件  共计：${sessionScope.gongji}元  您有预存款：0 元
        <br>
      </div>
    </td>
  </tr>
</table>
</form>
</body>
</html>
```

13.10　创建前台展示系统的 ActionForm

ActionForm 代表了由浏览器所传递过来的数据。每个 ActionForm 中定义的属性应该与页面中表单的控件或者说页面中所提交的数据相对应。这样，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。本节将创建网上书店管理应用系统的前台展示系统涉及的所有 ActionForm 类。所有的 ActionForm 类都继承于 org.apache.struts.validator.ValidatorForm 类。

13.10.1　创建图书搜索的 ActionForm

SearchActionForm 用于收集用户搜索图书的关键字信息。SearchActionForm 是典型的 JavaBean，包括搜索关键字、等级 ID 等信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。对于每个属性都包括 getter/setter 方法。

跟我做

在“shop”工程的“actionform”包中创建 SearchActionForm.java 类，编辑该文件，输入如下代码信息：

```
public class SearchActionForm extends ActionForm {
	// action 属性
	private String action;

	// searchkey 属性
	private String searchkey;

	// stepid 属性
	private String stepid;

	// typeid 属性
	private String typeid;

	/**
	 * action 属性的 getter 方法
	 *
	 * @return
	 */
	public String getAction() {
		return action;
	}

	/**
	 * action 属性的 setter 方法
	 *
	 * @param action
	 */
	public void setAction(String action) {
		this.action = action;
	}

	/**
	 * typeid 属性的 setter 方法
	 *
	 * @param typeid
	 */
	public void setTypeid(String typeid) {
		this.typeid = typeid;
	}

	/**
	 * stepid 属性的 setter 方法
	 *
	 * @param stepid
```

```
     */
    public void setStepid(String stepid) {
        this.stepid = stepid;
    }

    /**
     * searchkey 的 setter 方法
     *
     * @param searchkey
     */
    public void setSearchkey(String searchkey) {
        this.searchkey = searchkey;
    }

    /**
     * searchkey 的 getter 方法
     *
     * @return
     */
    public String getSearchkey() {
        return searchkey;
    }

    /**
     * stepid 的 getter 方法
     *
     * @return
     */
    public String getStepid() {
        return stepid;
    }

    /**
     * typeid 的 getter 方法
     *
     * @return
     */
    public String getTypeid() {
        return typeid;
    }

    public ActionErrors validate(ActionMapping actionMapping,
            HttpServletRequest httpServletRequest) {
        /** @todo: finish this method, this is just the skeleton. */
        return null;
    }

    public void reset(ActionMapping actionMapping,
            HttpServletRequest servletRequest) {
```

```
        }
}
```

13.10.2　创建购物车 ActionForm

ShopCarActionForm 是典型的 JavaBean，包括购买图书的名字、数量等信息，对于每个属性都有其对应的 getter/setter 方法。负责收集用户购买图书数量等信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。

跟我做

在“actionform”包中创建 ShopCarActionForm.java 类，编辑该文件，输入如下代码信息：

```
public class ShopCarActionForm extends ActionForm {
        // bookCompany 属性
        private String bookCompany;

        // bookCount 属性
        private String bookCount;

        // bookName 属性
        private String bookName;

        // bookPrice
        private String bookPrice;

        // bookType
        private String bookType;

        /**
         * bookCompany 的 getter 方法
         *
         * @return
         */
        public String getBookCompany() {
                return bookCompany;
        }

        /**
         * bookCompany 的 setter 方法
         *
         * @param bookCompany
         */
        public void setBookCompany(String bookCompany) {
                this.bookCompany = bookCompany;
        }
```

```
/**
 * bookType 的 setter 方法
 *
 * @param bookType
 */
public void setBookType(String bookType) {
	this.bookType = bookType;
}

/**
 * bookPrice 的 setter 方法
 *
 * @param bookPrice
 */
public void setBookPrice(String bookPrice) {
	this.bookPrice = bookPrice;
}

/**
 * bookName 的 setter 方法
 *
 * @param bookName
 */
public void setBookName(String bookName) {
	this.bookName = bookName;
}

/**
 * bookCount 的 setter 方法
 *
 * @param bookCount
 */
public void setBookCount(String bookCount) {
	this.bookCount = bookCount;
}

/**
 * bookCount 的 getter 方法
 *
 * @return
 */
public String getBookCount() {
	return bookCount;
}

/**
 * bookName 属性的 getter 方法
 *
 * @return
```

```
     */
    public String getBookName() {
        return bookName;
    }

    /**
     * bookPrice 的 getter 方法
     *
     * @return
     */
    public String getBookPrice() {
        return bookPrice;
    }

    /**
     * bookType 的 getter 方法
     *
     * @return
     */
    public String getBookType() {
        return bookType;
    }

    public ActionErrors validate(ActionMapping actionMapping,
            HttpServletRequest request) {
        /** @todo: finish this method, this is just the skeleton. */
        return null;
    }

    public void reset(ActionMapping actionMapping, HttpServletRequest request) {

    }
}
```

13.10.3 创建用户注册 ActionForm

RegActionForm 是典型的 JavaBean，包括用户名、密码和确认密码等信息。对于每个属性都有其对应的 getter/setter 方法。负责收集用户注册时的相关信息。通过在 Struts 配置文件中进行适当配置，Struts 框架会自动地将用户所提交的参数赋值到 ActionForm 中相应的属性中。

跟我做

在“actionform”包中创建 RegActionForm.java 类，编辑该文件，输入如下代码信息：

```
public class RegActionForm extends ActionForm {
    // realName 属性
    private String realName;
```

```
// userAdds 属性
private String userAdds;

// userAge 属性
private String userAge;

// userCard 属性
private String userCard;

// userCode 属性
private String userCode;

// userMail 属性
private String userMail;

// userName 属性
private String userName;

// userPhone 属性
private String userPhone;

// userPwd 属性
private String userPwd;

// userPwd1 属性
private String userPwd1;

// userSex 属性
private String userSex;

// userWork 属性
private String userWork;

// userCity 属性
private String userCity;

/**
 * realName 的 getter 方法
 *
 * @return
 */
public String getRealName() {
	return realName;
}

/**
 * realName 属性的 setter 方法
 *
```

```
	 * @param realName
	 */
	public void setRealName(String realName) {
		this.realName = realName;
	}

	/**
	 * userWork 属性的 setter 方法
	 *
	 * @param userWork
	 */
	public void setUserWork(String userWork) {
		this.userWork = userWork;
	}

	/**
	 * userSex 属性的 setter 方法
	 *
	 * @param userSex
	 */
	public void setUserSex(String userSex) {
		this.userSex = userSex;
	}

	/**
	 * userPwd1 的 setter 方法
	 *
	 * @param userPwd1
	 */
	public void setUserPwd1(String userPwd1) {
		this.userPwd1 = userPwd1;
	}

	/**
	 * userPwd 的 setter 方法
	 *
	 * @param userPwd
	 */
	public void setUserPwd(String userPwd) {
		this.userPwd = userPwd;
	}

	/**
	 * userPhone 的 setter 方法
	 *
	 * @param userPhone
	 */
	public void setUserPhone(String userPhone) {
		this.userPhone = userPhone;
```

```
}

/**
 * userName 的 setter 方法
 *
 * @param userName
 */
public void setUserName(String userName) {
        this.userName = userName;
}

/**
 * userMail 的 setter 方法
 *
 * @param userMail
 */
public void setUserMail(String userMail) {
        this.userMail = userMail;
}

/**
 * userCode 的 setter 方法
 *
 * @param userCode
 */
public void setUserCode(String userCode) {
        this.userCode = userCode;
}

/**
 * userCard 的 setter 方法
 *
 * @param userCard
 */
public void setUserCard(String userCard) {
        this.userCard = userCard;
}

/**
 * userAge 的 setter 方法
 *
 * @param userAge
 */
public void setUserAge(String userAge) {
        this.userAge = userAge;
}

/**
 * userAdds 的 setter 方法
```

```
 *
 * @param userAdds
 */
public void setUserAdds(String userAdds) {
        this.userAdds = userAdds;
}

/**
 * userCity 的 setter 方法
 *
 * @param userCity
 */
public void setUserCity(String userCity) {
        this.userCity = userCity;
}

/**
 * userAdds 的 getter 方法
 *
 * @return
 */
public String getUserAdds() {
        return userAdds;
}

/**
 * userAge 的 getter 方法
 *
 * @return
 */
public String getUserAge() {
        return userAge;
}

/**
 * userCard 的 getter 方法
 *
 * @return
 */
public String getUserCard() {
        return userCard;
}

/**
 * userCode 的 getter 方法
 *
 * @return
 */
public String getUserCode() {
```

```
		return userCode;
	}

	/**
	 * userMail 的 getter 方法
	 *
	 * @return
	 */
	public String getUserMail() {
		return userMail;
	}

	/**
	 * userName 的 getter 方法
	 *
	 * @return
	 */
	public String getUserName() {
		return userName;
	}

	/**
	 * userPhone 的 getter 方法
	 *
	 * @return
	 */
	public String getUserPhone() {
		return userPhone;
	}

	/**
	 * userPwd 的 getter 方法
	 *
	 * @return
	 */
	public String getUserPwd() {
		return userPwd;
	}

	/**
	 * userPwd1 的 getter 方法
	 *
	 * @return
	 */
	public String getUserPwd1() {
		return userPwd1;
	}

	/**
```

```
	 * userSex 的 getter 方法
	 *
	 * @return
	 */
	public String getUserSex() {
		return userSex;
	}

	/**
	 * userWork 的 getter 方法
	 *
	 * @return
	 */
	public String getUserWork() {
		return userWork;
	}

	/**
	 * userCity 的 getter 方法
	 *
	 * @return
	 */
	public String getUserCity() {
		return userCity;
	}

	public ActionErrors validate(ActionMapping actionMapping,
			HttpServletRequest httpServletRequest) {
		/** @todo: finish this method, this is just the skeleton. */
		return null;
	}

	public void reset(ActionMapping actionMapping,
			HttpServletRequest servletRequest) {
	}
}
```

13.11　实现前台展示系统的控制层

当用户在页面中提交表单后，Struts 框架就会把用户请求转发给 Action 组件。本节实现网上书店前台展示系统中的所有 Action 类，实现其控制层，每个 Action 都覆盖了 Action 类的 execute()方法，在 execute()方法中从 request 方法取得必要的 page 属性，完成相应的业务逻辑。

跟我做

（1）在“shop”工程的“action”包中创建 PagerAction.java 类，编辑该文件，输入如下代码信息：

```
public class PagerAction extends Action {
    /**
     * 覆盖 Action 的 execute()方法
     */
    public ActionForward execute(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response) {
        // 从 request 中取得 page 属性
        String page = request.getParameter("page");
        // 从 request 中取得 TypeId 属性
        String bookTypeId = request.getParameter("TypeId");
        // 从 request 中取得 bookStep 属性
        String bookStep = request.getParameter("bookStep");
        // 创建 CustomerOperation 对象
        db.CustomerOperation md = new CustomerOperation();
        int countPage = 0;
        if (bookTypeId != null) {
            // 通过 CustomerOperation 对象取得页码
            countPage = md.getPageCount(1, 6, Integer.parseInt(bookTypeId),
                    Integer.parseInt(bookStep));
            // 通过 CustomerOperation 对象取得图书信息
            ArrayList bookList = md.getPageData(1, 6, Integer
                    .parseInt(bookTypeId), Integer.parseInt(bookStep));
            // 将查询结果存入到 request 中
            request.setAttribute("bookList", bookList);
            request.setAttribute("page", 1);
            request.setAttribute("TypeId", bookTypeId);
            request.setAttribute("countPage", countPage);
            request.setAttribute("bookStep", bookStep);
        }

        if (request.getParameter("next") != null) {
            String flage = request.getParameter("next");
            if (flage.equals("true")) {
                // 通过 CustomerOperation 对象取得页码
                countPage = md.getPageCount(1, 6, Integer.parseInt(bookTypeId),
                        Integer.parseInt(bookStep));
                // 通过 CustomerOperation 对象取得图书信息
                ArrayList bookList = md.getPageData(Integer.parseInt(page), 6,
                        Integer.parseInt(bookTypeId), Integer
                                .parseInt(bookStep));
                // 将查询结果存入到 request 中
                request.setAttribute("bookList", bookList);
                request.setAttribute("page", page);
                request.setAttribute("TypeId", bookTypeId);
```

```
                request.setAttribute("countPage", countPage);
                request.setAttribute("bookStep", bookStep);
            } else {
                // 通过 CustomerOperation 对象取得页码
                countPage = md.getPageCount(1, 6, Integer.parseInt(bookTypeId),
                        Integer.parseInt(bookStep));
                // 通过 CustomerOperation 对象取得图书信息
                ArrayList bookList = md.getPageData(Integer.parseInt(page), 6,
                        Integer.parseInt(bookTypeId), Integer
                                .parseInt(bookStep));
                // 将查询结果存入到 request 中
                request.setAttribute("bookList", bookList);
                request.setAttribute("page", page);
                request.setAttribute("TypeId", bookTypeId);
                request.setAttribute("countPage", countPage);
                request.setAttribute("bookStep", bookStep);

            }
        }

        if (page != null) {
            if (page.equals("frest")) {
                // 通过 CustomerOperation 对象取得页码
                countPage = md.getPageCount(1, 6, Integer.parseInt(bookTypeId),
                        Integer.parseInt(bookStep));
                // 通过 CustomerOperation 对象取得图书信息
                ArrayList bookList = md.getPageData(1, 6, Integer
                        .parseInt(bookTypeId), Integer.parseInt(bookStep));
                // 将查询结果存入到 request 中
                request.setAttribute("bookList", bookList);
                request.setAttribute("page", 1);
                request.setAttribute("TypeId", bookTypeId);
                request.setAttribute("countPage", countPage);
                request.setAttribute("bookStep", bookStep);
            }
            if (page.equals("last")) {
                // 通过 CustomerOperation 对象取得页码
                countPage = md.getPageCount(1, 6, Integer.parseInt(bookTypeId),
                        Integer.parseInt(bookStep));
                // 通过 CustomerOperation 对象取得图书信息
                ArrayList bookList = md.getPageData(countPage, 6, Integer
                        .parseInt(bookTypeId), Integer.parseInt(bookStep));
                // 将查询结果存入到 request 中
                request.setAttribute("bookList", bookList);
                request.setAttribute("page", countPage);
                request.setAttribute("TypeId", bookTypeId);
                request.setAttribute("countPage", countPage);
                request.setAttribute("bookStep", bookStep);
```

```
                }
            }

            return mapping.findForward("list");
        }
}
```

该 Action 实现了分页显示所有的图书信息。通过 CustomerOperation 对象获得特定页码的图书信息并将查询结果保存到 request 对象中。

（2）在“action”包中创建 RegAction.java 类，编辑该文件，输入如下代码信息：

```
public class RegAction extends Action {
        /**
         * 覆盖 Action 的 execute()方法
         */
        public ActionForward execute(ActionMapping mapping, ActionForm form,
                        HttpServletRequest request, HttpServletResponse response) {
                //将 form 参数进行正确的类型转换
                RegActionForm regForm = (RegActionForm) form;
                //创建 CustomerOperation 对象
                CustomerOperation md = new CustomerOperation();
                //创建 UserBean 对象
                UserBean userBean = new UserBean();
                //为 userBean 设置对应的属性
                userBean.setRealName(regForm.getRealName());
                userBean.setUserAdds(regForm.getUserAdds());
                userBean.setUserAge(Integer.parseInt(regForm.getUserAge()));
                userBean.setUserCard(regForm.getUserCard());
                userBean.setUserCity(Integer.parseInt(regForm.getUserCity()));
                userBean.setUserCode(Integer.parseInt(regForm.getUserCode()));
                userBean.setUserMail(regForm.getUserMail());
                userBean.setUserName(regForm.getUserName());
                userBean.setUserPhone(regForm.getUserPhone());
                userBean.setUserPwd(regForm.getUserPwd());
                userBean.setUserSex(Integer.parseInt(regForm.getUserSex()));
                userBean.setUserWork(regForm.getUserWork());
                //通过 CustomerOperation 对象插入新的用户信息
                if (md.insertUser(userBean)) {
                        request.setAttribute("regok", 1);
                        System.out.println("ok" + "    " + request.getAttribute("regok"));
                } else {
                        request.setAttribute("regok", 0);
                        System.out.println("no" + "    " + request.getAttribute("regok"));
                }
                return mapping.findForward("reg");
        }
}
```

该 Action 完成了用户的注册功能。从 RegActionForm 中读取所有的用户注册信息，然

后通过 CustomerOperation 对象将这些信息保存到数据库中。

（3）在“action”包中创建 LoginAction.java 类，编辑该文件，输入如下代码信息：

```
public class LoginAction extends Action {
	/**
	 * 覆盖 Action 的 execute()方法
	 *
	 */
	public ActionForward execute(ActionMapping mapping, ActionForm form,
			HttpServletRequest request, HttpServletResponse response) {
		// 将 form 对象进行适当的类型转换
		LoginActionForm loginForm = (LoginActionForm) form;
		// 从 actionform 中取得用户名和用户密码
		String userName = loginForm.getUserNmae();
		String userPwd = loginForm.getUserPwd();
		// 创建 CustomerOperation 对象
		CustomerOperation md = new CustomerOperation();
		if (request.getParameter("action") == null) {
			if (userName.equals("") || userPwd.equals("")) {
				return mapping.findForward("index");
			}
			// 通过 CustomerOperation 对象验证登录的用户
			UserBean useBean = md.checkUsersLogin(userName, userPwd);
			if (useBean != null) {
				md.Close();
				// 将特定信息保存到 request 中
				request.getSession().setAttribute("useBean", useBean);
				request.getSession().setAttribute("shangpin", "0");
				request.getSession().setAttribute("zongshu", "0");
				request.getSession().setAttribute("gongji", "0");
				request.getSession().setAttribute("loginms", 1);
				request.getSession().setAttribute("list",
						new java.util.ArrayList());
				return mapping.findForward("index");
			} else {
				request.getSession().setAttribute("useBean", useBean);
				request.setAttribute("loginms", 0);
				md.Close();
				return mapping.findForward("index");
			}

		}
		if (request.getParameter("action").equals("out")) {
			request.getSession().removeAttribute("useBean");
			request.getSession().removeAttribute("list");
			request.getSession().removeAttribute("shangpin");
			request.getSession().removeAttribute("zongshu");
			request.getSession().removeAttribute("gongji");
			return mapping.findForward("index");
```

```
            }
            return mapping.findForward("login");
        }
}
```

该 Action 完成了用户的登录。通过 CustomerOperation 对象完成用户的验证。

13.12　生成 Struts 的配置文件

Struts 的核心是控制器，即 ActionServlet，而 ActionServlet 的核心就是 Struts-config.xml，Struts-config.xml 集中了所有页面的导航定义。掌握 Struts-config.xml 是掌握 Struts 的关键所在。本节创建 Struts 的配置文件，将 JSP 页面、Action 等组件结合起来。

跟我做

在“shop”工程中创建“struts-config.xml”文件，编辑该文件，在文件中输入如下代码：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">

<struts-config>
<!--FromBean 的定义-->
  <form-beans>
    <!--定义名字为 loginActionForm 的 form-bean-->
    <form-bean
        name="loginActionForm"
        type="form.reg.LoginActionForm" />
    <!--定义名字为 bookInfActionForm 的 form-bean-->
    <form-bean
        name="bookInfActionForm"
        type="actionform.bookinf.BookInfActionForm" />
     <!--定义名字为 shopCarActionForm 的 form-bean-->
    <form-bean
        name="shopCarActionForm"
        type="actionform.ShopCarActionForm" />
     <!--定义名字为 indexActionForm 的 form-bean-->
    <form-bean
        name="indexActionForm"
        type="action.IndexActionForm" />
    <!--定义名字为 formBean 的 form-bean-->
    <form-bean
        name="formBean"
        type="action.IndexActionForm" />
     <!--定义名字为 regActionForm 的 form-bean-->
    <form-bean
        name="regActionForm"
        type="form.reg.RegActionForm" />
```

```
    <!--定义名字为 searchActionForm 的 form-bean-->
   <form-bean
      name="searchActionForm"
      type="actionform.SearchActionForm" />
    <!--定义名字为 editUserForm 的 form-bean-->
   <form-bean
      name="editUserForm"
      type="actionform.EditUserForm" />
</form-beans>
 <!--定义全局页面的跳转-->
<global-forwards>
   <!--定义名字为 index 的全局跳转-->
   <forward name="index" path="/jsp/index1.jsp" />
    <!--定义名字为 adminIndex 的全局跳转-->
   <forward name="adminIndex" path="/admin/index.jsp" />
    <!--定义名字为 login 的全局跳转-->
   <forward name="login" path="/jsp/reg/login1.jsp" />
    <!--定义名字为 shopcar 的全局跳转-->
   <forward name="shopcar" path="/jsp/shopcar.jsp" redirect="false" />
    <!--定义名字为 list 的全局跳转-->
   <forward name="list" path="/jsp/list.jsp" />
    <!--定义名字为 search 的全局跳转-->
   <forward name="search" path="/jsp/search.jsp" redirect="false" />
    <!--定义名字为 reg 的全局跳转-->
   <forward name="reg" path="/jsp/reg/reg.jsp" redirect="false" />
    <!--定义名字为 userlist 的全局跳转-->
   <forward name="userlist" path="/admin/userList.jsp" />
    <!--定义名字为 orderlist 的全局跳转-->
   <forward name="orderlist" path="/admin/orderList.jsp" />
    <!--定义名字为 booklist 的全局跳转-->
   <forward name="booklist" path="/admin/bookList.jsp" />
    <!--定义名字为 bookadd 的全局跳转-->
   <forward name="bookadd" path="/admin/books_add.jsp" />
    <!--定义名字为 booktype 的全局跳转-->
   <forward name="booktype" path="/admin/booktype.jsp" />
    <!--定义名字为 bookedit 的全局跳转-->
   <forward name="bookedit" path="/admin/books_edit.jsp" />
    <!--定义名字为 BookPage 的全局跳转-->
   <forward name="BookPage" path="/admin/BookPage.do?page=1" />
    <!--定义名字为 UserPage 的全局跳转-->
   <forward name="UserPage" path="/admin/UserPage.do?page=1" />
    <!--定义名字为 addadmin 的全局跳转-->
   <forward name="addadmin" path="/admin/add_admin.jsp" />
    <!--定义名字为 changeadmin 的全局跳转-->
   <forward name="changeadmin" path="/admin/chengAdmin.jsp" />
</global-forwards>
<action-mappings>
   <!-- 定义 path 为"/ loginAction"的 Action -->
   <action       input="/jsp/reg/login.jsp"       name="loginActionForm"       path="/loginAction"
```

```
scope="request" type="action.reg.LoginAction" validate="true" />
        <!-- 定义 path 为"/ bookInfAction"的 Action -->
     <action    input="/jsp/bookinf.jsp"    name="bookInfActionForm"    path="/bookInfAction"
scope="session" type="action.bookinf.BookInfAction" validate="true" />
        <!-- 定义 path 为"/ pagerAction"的 Action -->
     <action name="formBean" path="/pagerAction" type="action.PagerAction" />
        <!-- 定义 path 为"/ regAction"的 Action -->
     <action  input="/jsp/reg/reg.jsp"  name="regActionForm"  path="/regAction"  scope="request"
type="action.RegAction" validate="true" />
        <!-- 定义 path 为"/jsp/orderInsertAction"的 Action -->
     <action path="/jsp/orderInsertAction" type="action.OrderInsertAction" />
     <action path="/admin/adminLoginAction" type="action.AdminLoginAction" />
     <action path="/admin/UserPage" type="action.UserPage" />
     <action path="/admin/OrderPage" type="action.OrderPage" />
     <action path="/admin/BookPage" type="action.BookPage" />
     <action path="/admin/InsertBook" type="action.InsertBook" />
     <action path="/admin/InsertBookType" type="action.InsertBookType" />
     <action path="/admin/DeleteBookType" type="action.DeleteBookType" />
     <action path="/admin/UpdateBookType" type="action.UpdateBookType" />
     <action    input="/admin/user_edit.jsp"    name="editUserForm"    path="/admin/EditUser"
scope="request" type="action.EditUser" validate="true" />
     <action path="/admin/InsertAdmin" type="action.InsertAdmin" />
     <action path="/admin/ChengeAdminPwd" type="action.ChengeAdminPwd" />
     <action path="/admin/AdminLoginOut" type="action.AdminLoginOut" />
   </action-mappings>
   <message-resources parameter="ApplicationResources" />
</struts-config>
```

该配置文件是 Struts 的核心，将 JSP 页面、Action 等组件结合起来。

第 14 章　综合实例——餐费管理系统

前面的章节介绍了如何使用 Eclipse，本章将以一个综合应用实例的形式来进一步了解在 Eclipse 中进行应用开发的过程。这个实例将用到一些目前比较流行的开源框架，如 Strut、Spring、Hibernate，Eclipse 对这些框架都有很好的支持。

14.1　项目需求分析

项目需求分析是软件开发的第一步，需求分析的目的是对系统进行评估，采集和分析系统的需求，理解系统要解决的问题，充分考虑系统的实用性，并建立相应的业务模型。

14.1.1　需求概述

- 项目名称：餐费管理系统（Repast Expenses Management）。
- 项目需求：公司内部餐厅使用计算机进行餐费管理，即每次员工就餐时在线刷卡（将自己的登录号码输入到餐费管理系统中），系统将从该名员工的账户中扣除本次就餐的费用（就餐种类主要是套餐，套餐有不同的收费标准，员工可以自己选择），如果员工账户余额不足，允许透支，但最多透支 3 次。如果员工发现自己的余额不足，应该及时补充余额。否则透支超过 3 次，将取消该员工的就餐权利。

14.1.2　功能模块需求分析

通过 14.1.1 小节的需求概述，得知了用户的具体业务需求，根据这些需求可以获知系统中所要用到的功能模块，如图 14-1 所示。

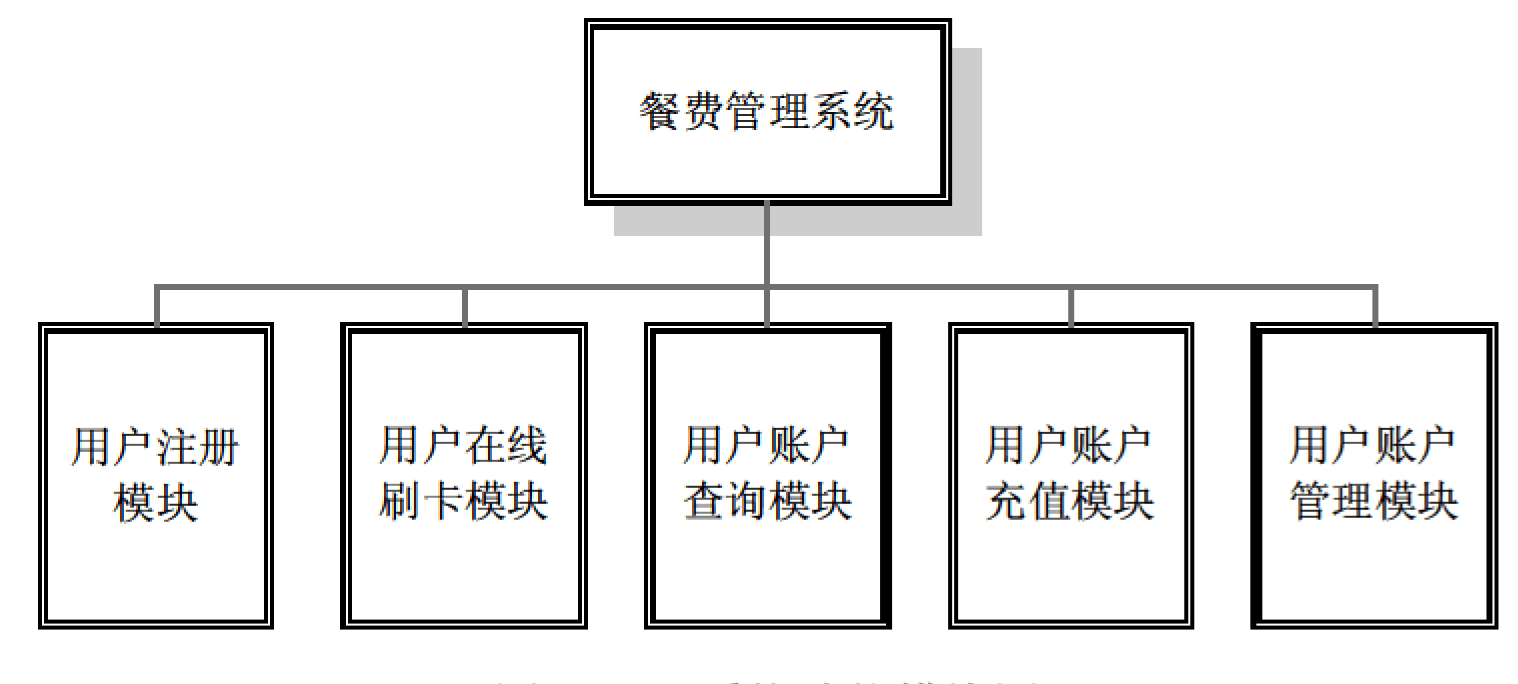


图 14-1　系统功能模块图

- 用户注册模块：用户在系统注册，获得就餐卡号（登录号）。注册时需要先输入员工的姓名和身份证号码，系统从公司员工数据库（假设已经存在）中查询公司是否存在该员工，如果存在允许进行注册。否则不允许注册。注册时一个员工只允许注册一个账号。
- 用户在线刷卡模块：用户在就餐时刷卡（将自己的就餐卡号输入到餐费管理系统中），系统判断员工的账户余额，并从账户余额中减去本次就餐的费用，并显示本次就餐发生费用和用户的账户余额。
- 用户账户查询模块：查询员工的就餐费用的历史记录以及余额的情况。
- 员工账户充值模块：管理员收到员工的餐费后对用户的账户余额进行充值，即把收到的餐费和员工账户的余额进行累加。
- 员工账户管理模块：可以对员工的账户进行删除、修改。

14.1.3　用例需求分析

通过 14.1.1 小节的需求概述，还获知了用户的具体业务需求以及系统中存在的角色。下面将根据功能需求和角色建立业务模型，通过模型来描述用户的业务需求，同时也为下一步的分析和设计做好了准备工作。在需求分析时，用例图（Use Case）是一种常用的建立业务模型的方式。根据需求，可以得到产品信息反馈平台的业务用例图，如图 14-2 所示。

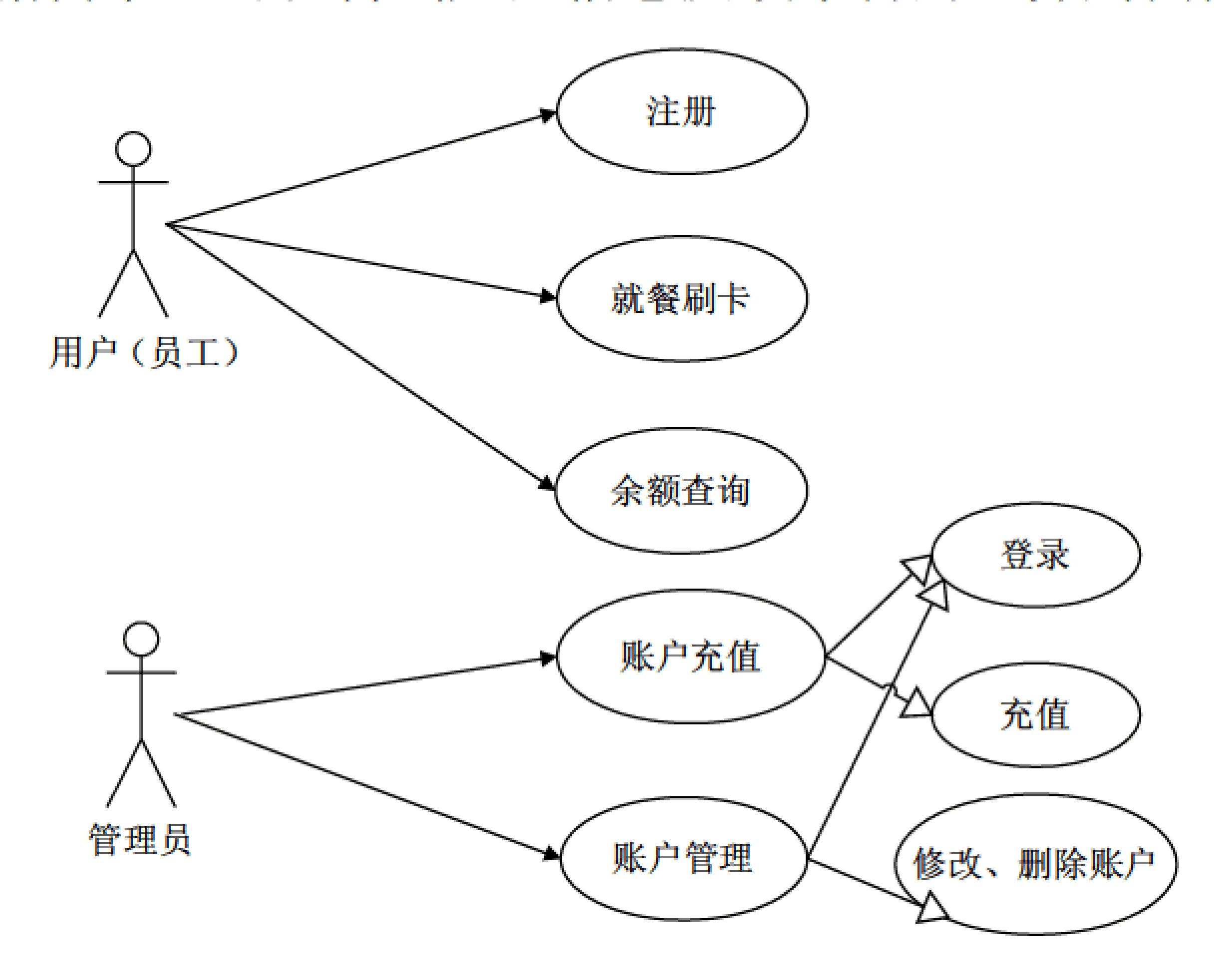


图 14-2　餐费管理系统用例图

从图 14.2 中可以看到这个产品信息反馈平台系统中主要有两个角色，分别是员工和餐费管理员。这两个角色分别进行了不同的操作，根据这些操作建立相应的用例。系统中共有 5 个用例，分别是：

- 员工就餐账户注册用例。
- 员工刷卡就餐用例。
- 员工查询账户余额用例。
- 就餐账户充值用例。

- ❑ 员工账户管理用例。

下面几节分别对这几个用例进行介绍。

14.1.4　员工就餐账户注册用例

员工进行就餐账户的注册页面，如图 14-3 所示。

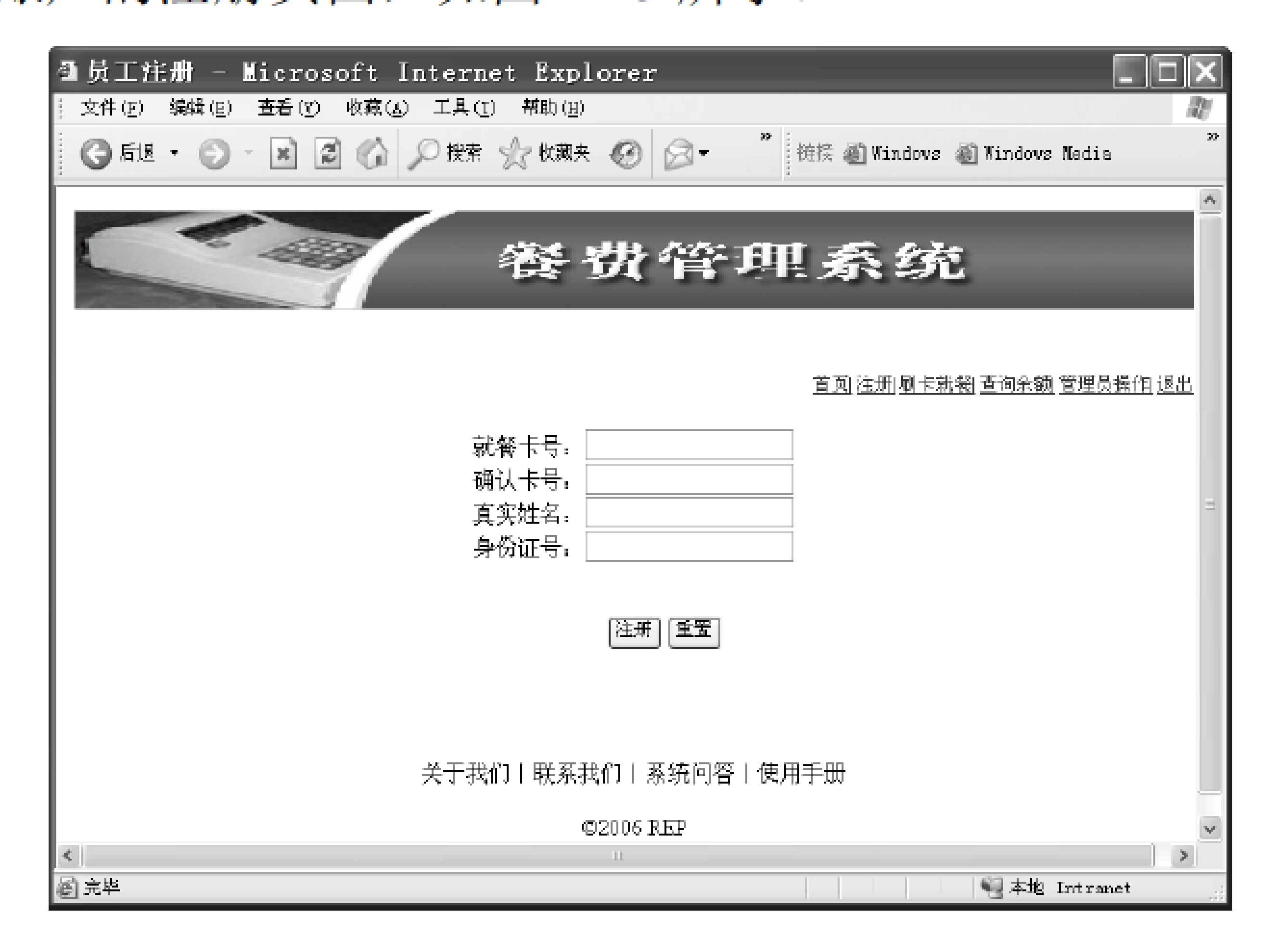


图 14-3　员工注册页面

员工输入就餐卡号，再次确认后输入员工的姓名和身份证号，系统将到公司员工数据库中去查找是否存在该名员工，如果存在则要查找账户是否存在，如果不存在这个账户，则允许注册。

14.1.5　员工刷卡就餐用例

员工刷卡就餐（这里的卡并非真正意义上的卡片，而是用户注册后建立的账户名称）页面，如图 14-4 所示。



图 14-4　员工刷卡就餐页面

员工输入就餐的卡号，也就是账户名称，然后输入本次就餐的标准，系统将检查员工的就餐账户是否透支超过 3 次，超过 3 次则不允许就餐。

14.1.6　员工查询账户余额用例

员工查询就餐账户的余额：员工可以在任何时候在线查看自己账户的余额，如图 14-5 所示。



图 14-5　查询余额

员工输入就餐卡号后，系统将显示出员工就餐账户的余额以及透支的次数。

14.1.7　就餐账户充值用例

员工账户充值由餐费管理员完成，管理员根据员工缴纳的费用来进行充值，在利用系统进行充值之前，管理员需要登录，如图 14-6 所示。

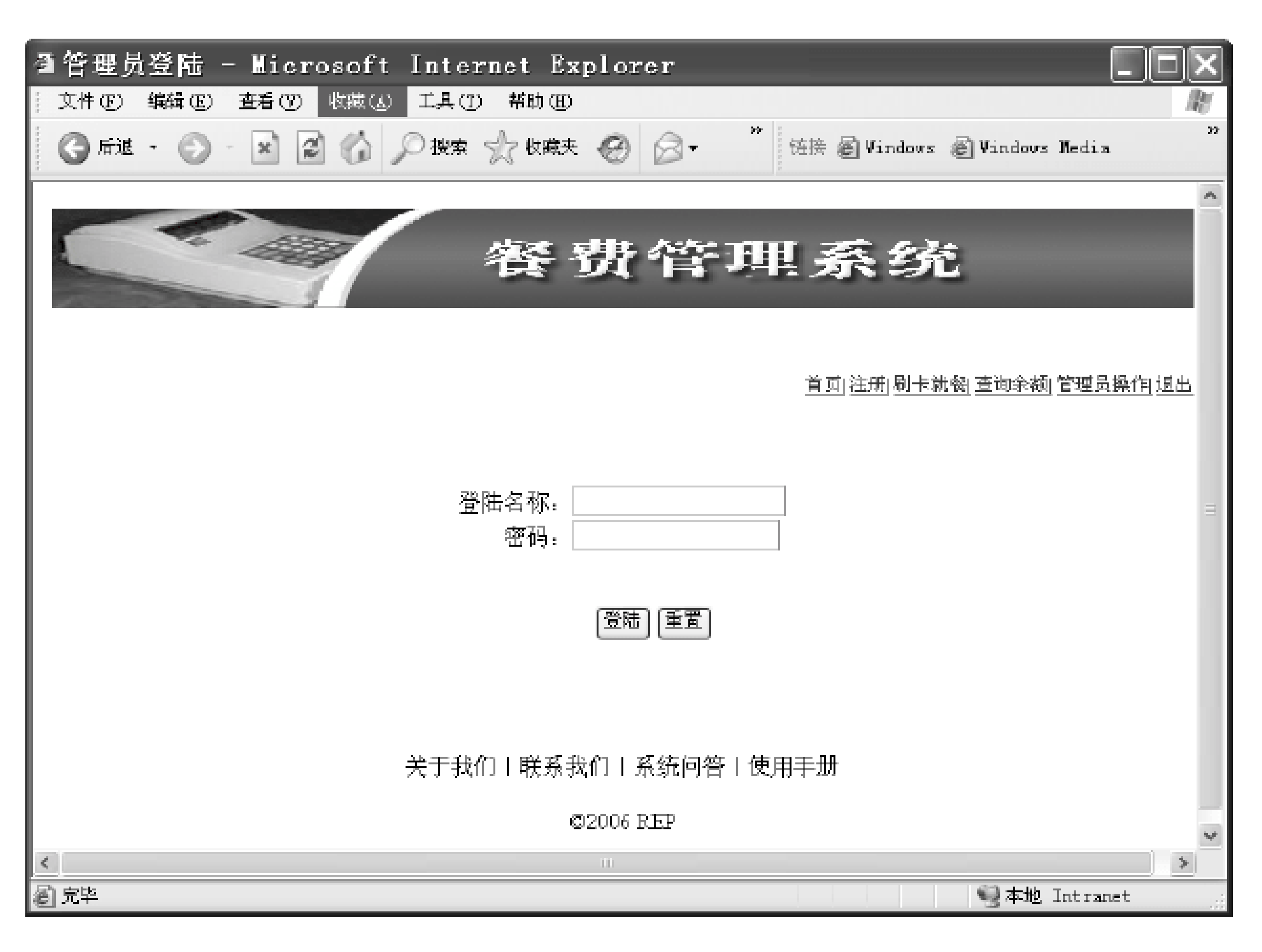


图 14-6　管理员登录

登录验证通过后，进入管理员操作页面，如图 14-7 所示。

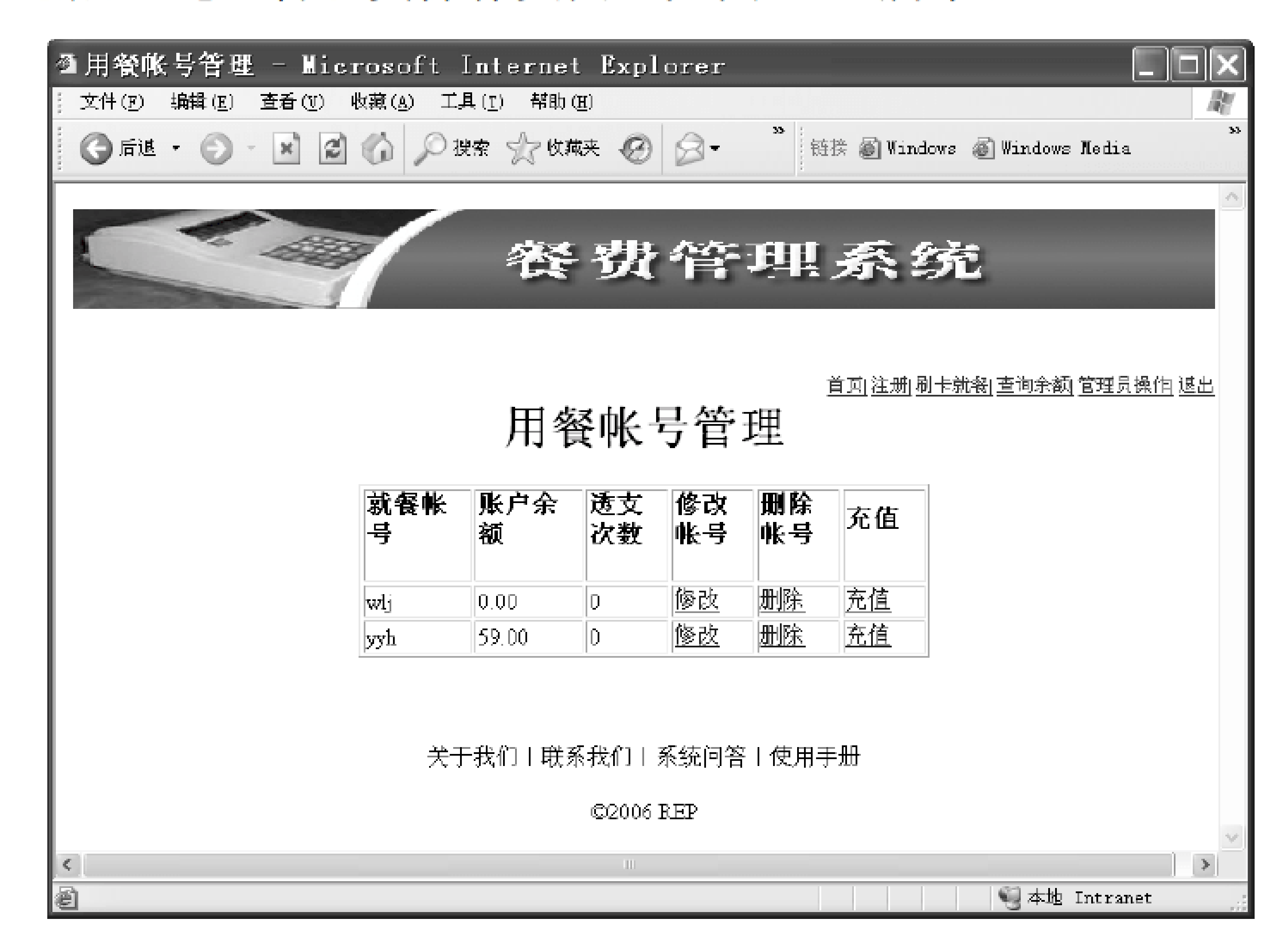


图 14-7　管理员操作页面

单击“充值”超链接后，进入账号充值页面，如图 14-8 所示。



图 14-8　就餐账户充值页面

14.1.8　员工账户管理用例

员工账户管理用例主要进行账户名称的修改和删除。在进行账户管理操作之前，首先要进行登录验证，如图 14-6 所示。验证通过后进入管理员操作页面，单击“修改”或“删除”超链接即可进行相应的操作，如图 14-9 所示。



图 14-9　就餐账号修改

14.2　系统分析和设计

在需求分析的基础上可以进行系统分析和设计，系统分析和设计主要进行数据库的分析和设计，以及和业务相关的主要类和接口的设计。

14.2.1　数据库分析和设计

根据需求，在产品信息反馈平台中涉及的实体主要有员工、员工账户和管理员。

- 员工：在公司餐厅就餐的公司职工。
- 员工账户：员工就餐使用的账户。
- 管理员：对员工账户进行管理的人员。

对实体进行了深入的分析后，可以确定实体之间的关系，如图 14-10 所示。

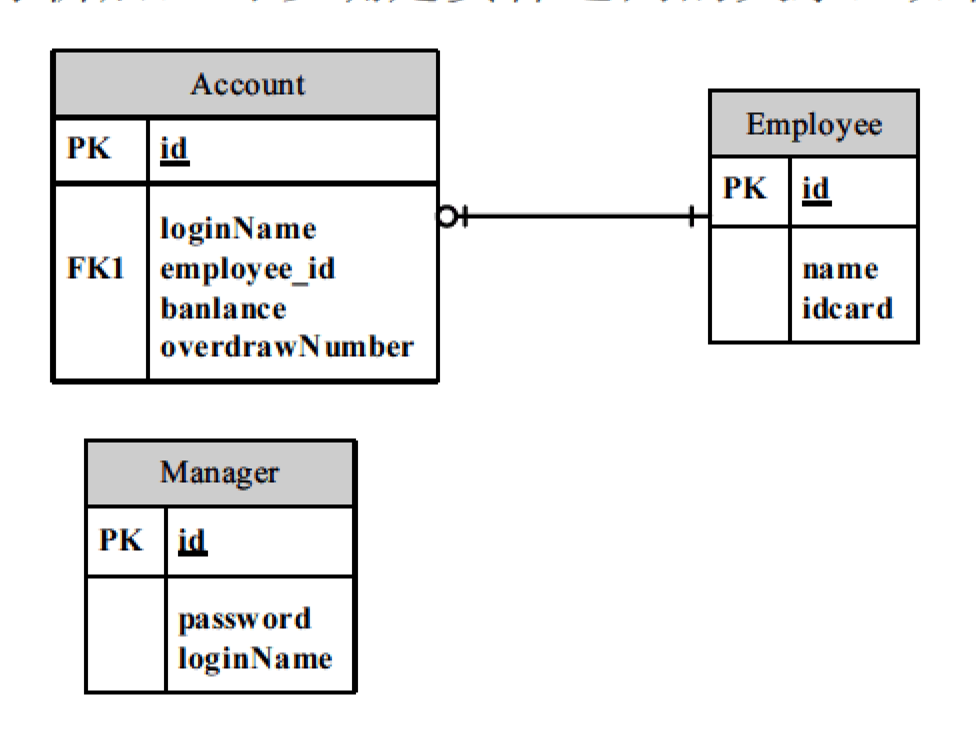


图 14-10　产品信息反馈平台 ER 图

产品信息反馈平台 ER 图表示了系统中实体之间的关系，如一个员工可以有零个或一个

账户，一个账户必须只能属于一个员工，因此员工和账户之间是一对一的关系。

其中 Manager 从严格意义上说并不能算作实体，因为在数据库中管理员（Manager）表只有预先设定好的一条记录，在应用中没有数据的改变，这也就不符合实体的定义，但是在这里为了使读者对系统有个整体的认识，因此也把 Manager 列在这里，但是没有和其他实体产生关系。

根据 ER 图，确定了各个实体间的关系和属性后，可以在数据库中建立两个表，即 Account 表和 Manager 表，因为员工数据库是从公司的其他系统中获得，因此不需要重新建立，但是为了使读者更容易理解，这里将这 3 个表的相关字段和相应的 DDL 脚本分别给予介绍。具体表的结构和建立脚本如下。

- Employee（员工）表结构，如表 14-1 所示：

表 14-1　Employee（员工）表结构

别　　名	类　　型	描　　述
id	int（11）	用户标示，自增主键
name	varchar（20）	用户姓名
idcard	varchar（10）	身份证号

建立脚本如下：

```
CREATE TABLE 'employee' (
  'id' int(11) NOT NULL auto_increment,
  'name' varchar(20) NOT NULL default '',
  'idcard' varchar(18) NOT NULL default '',
  PRIMARY KEY   ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- Account（就餐账户）表结构，如表 14-2 所示：

表 14-2　Account（就餐账户）表结构

别　　名	类　　型	描　　述
id	int（11）	员工就餐账户标示，是自增主键
loginName	varchar（20）	员工就餐账户名称
banlance	double（5,2）	员工就餐账户余额
overdrawNumber	int（1）	员工就餐透支次数
employee_id	int（11）	账户所对应的员工标示

建立 Account 表的 DDL 脚本如下：

```
CREATE TABLE 'account' (
  'id' int(11) NOT NULL auto_increment,
  'loginName' varchar(20) NOT NULL default '',
  'balance' double(5,2) default '0.00',
  'overdrawNumber' int(1) unsigned zerofill default '0',
```

```
  'employee_id' int(11) NOT NULL default '0',
  PRIMARY KEY   ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

- ❑ Manager（管理员）表结构，如表 14-3 所示：

表 14-3　Manager（管理员）表结构

别　　名	类　　型	描　　述
id	int(11)	产品经理标示，是自增主键
loginName	varchar(20)	管理员登录名

建立脚本如下：

```
CREATE TABLE 'productmanager' (
  'id' int(11) NOT NULL auto_increment,
  'loginName' varchar(20) NOT NULL default '',
  'password' varchar(10) NOT NULL default '',
   PRIMARY KEY   ('id')
   UNIQUE KEY 'login' ('loginName')
);
```

- ❑ Product（产品）表结构，如表 14-4 所示：

表 14-4　Product（产品）表结构

别　　名	类　　型	描　　述
id	int(11)	产品标示，是自增主键
name	varchar(20)	产品的名字

建立脚本如下：

```
CREATE TABLE 'manager' (
  'id' int(11) NOT NULL auto_increment,
  'loginName' varchar(20) NOT NULL default '',
  'password' varchar(20) NOT NULL default '',
  PRIMARY KEY   ('id')
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

14.2.2　业务逻辑层和 DAO 层设计

在这个应用实例中采用的是 B/S 4 层架构，包括表示层、业务逻辑层、数据访问层、数据层。

- ❑ 表示层主要采用的是 Struts 构架，Struts 框架实现了 MVC 模型，使得显示、控制和模型部分相互分离，提高了代码的可重用性。表示层中的 Struts 框架实现了 MVC 模型中的视图部分和控制部分。MVC 中的模型部分主要分布在业务逻辑层。

- 业务逻辑层主要使用 Spring 框架来实现，Spring 框架使用依赖注入的方式，使得业务逻辑组件在运行期被注入到容器中，提高系统的可维护性，而且利用 Spring 框架的 AOP（面向方面的编程）功能可以从面向方面的角度更好地降低系统中各个组件之间的耦合性，Spring 的事务管理功能也是 Spring 框架的一个重要内容。
- 数据访问层，又称 DAO 层，在这层主要完成对象-关系映射的建立，通过这个映射，可以通过访问业务对象即可实现对数据库的访问，使得开发中不必再用 SQL 语句编写复杂的数据库访问程序，这样就简化了对数据库访问，提高了开发效率。同时通过对象-关系映射的配置，可以建立业务对象之间的复杂关系，如一对多、多对一、一对一、多对多等。这样就不再需要在数据库中建立表之间的复杂联系，使得业务对象之间的关系和数据库相分离，简化了数据库的建立和维护。在这一层中主要使用 Hibernate 框架来实现。
- 数据层，主要是数据库，本系统中使用的是 MySql 数据库。

4 层构架如图 14-11 所示。

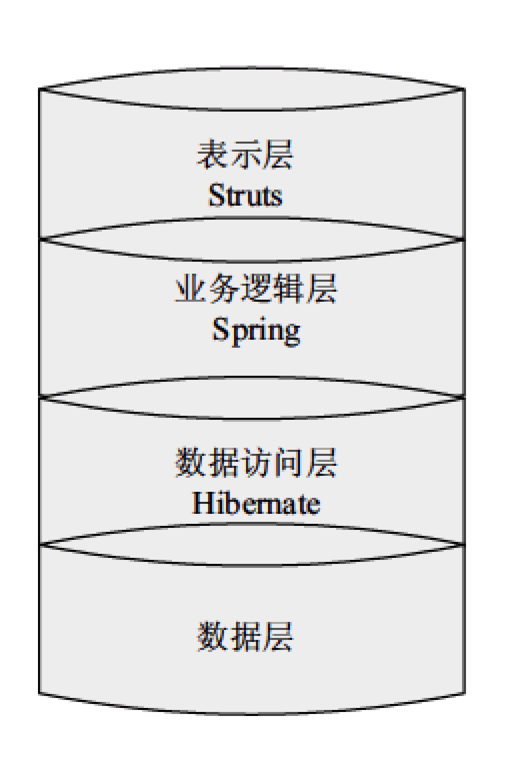


图 14-11　4 层 B/S 结构

在业务逻辑层中主要包含的内容有业务实体、业务逻辑等内容。图 14-12 是业务逻辑层的类图。从图中可以看到主要业务逻辑包括与管理员相关的业务逻辑：ManagerServiceImpl 类，以及和员工相关的业务逻辑：EmployeeServiceImpl 类，这些类分别实现了相应的接口，这些类在工作的过程中还需要依赖相应的 DAO 对象和 JavaBean 对象。

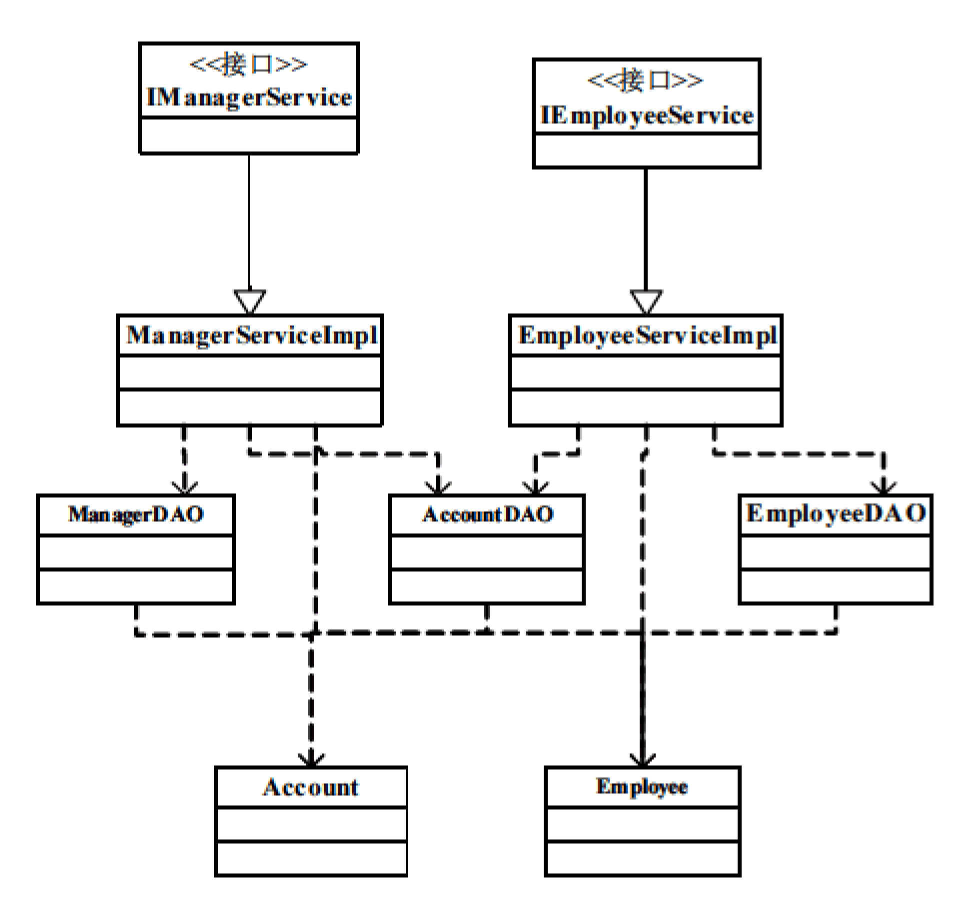


图 14-12　业务逻辑类图

数据访问层主要完成对数据库的访问，图 14-13 表示了 DAO 层的类图。从图中可以看到，系统要用到的 DAO 类有 ManagerDAO、AccountDAO、EmployeeDAO，这些 DAO 分别实现了各自的接口，同时都继承了_BaseRootDAO 类，_BaseRootDAO 类中包含了大量常规的对数据进行访问的方法，_BaseRootDAO 类实现了抽象类_RootDAO。

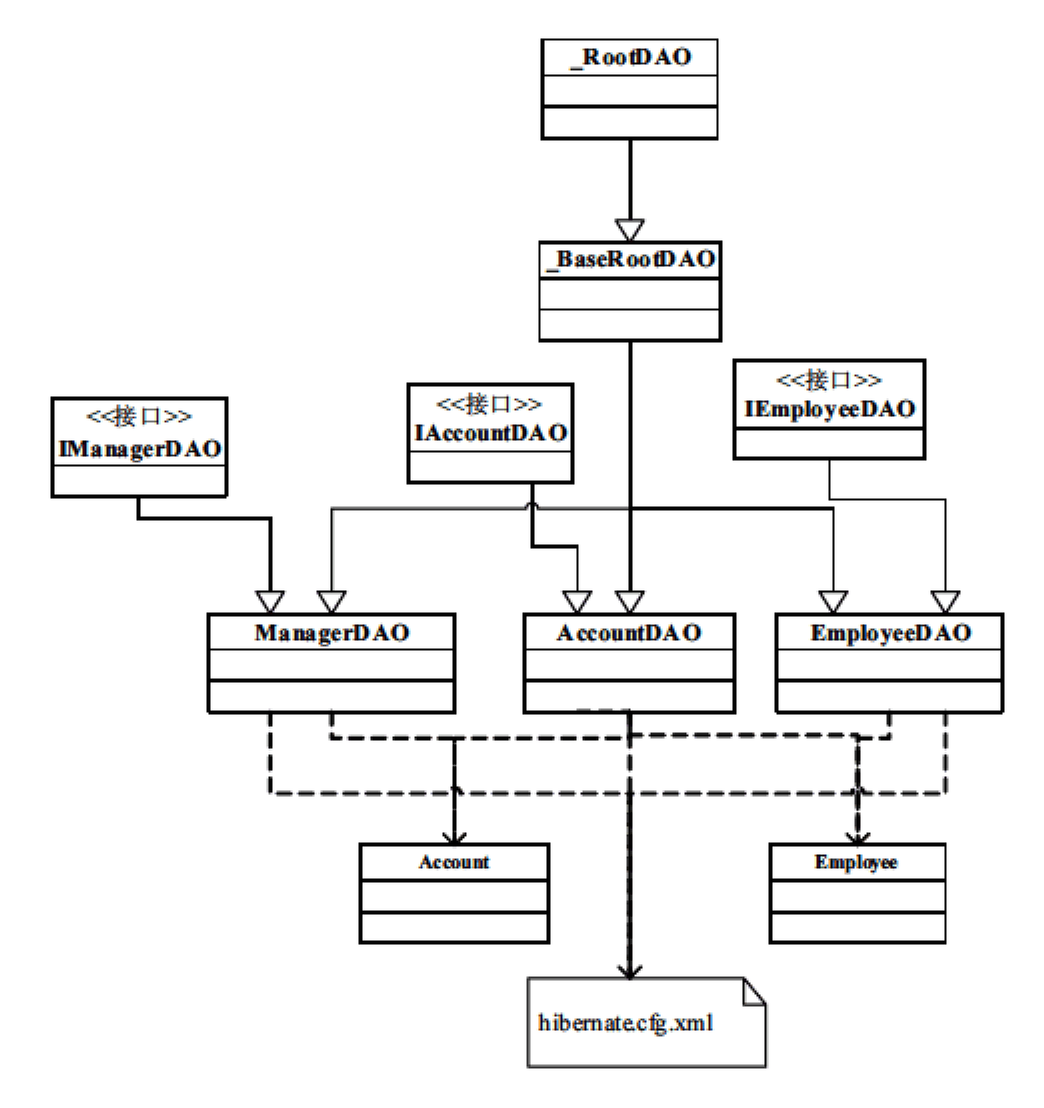


图 14-13　DAO 层类图

同时还可以看到这些 DAO 根据 Hibernate 配置文件可以访问不同的对象-关系映射文件，这样在 DAO 类中就可以通过对业务实体对象的操作来实现相应数据库的操作。

14.2.3　系统的包

系统中需要建立的包如图 14-14 所示。

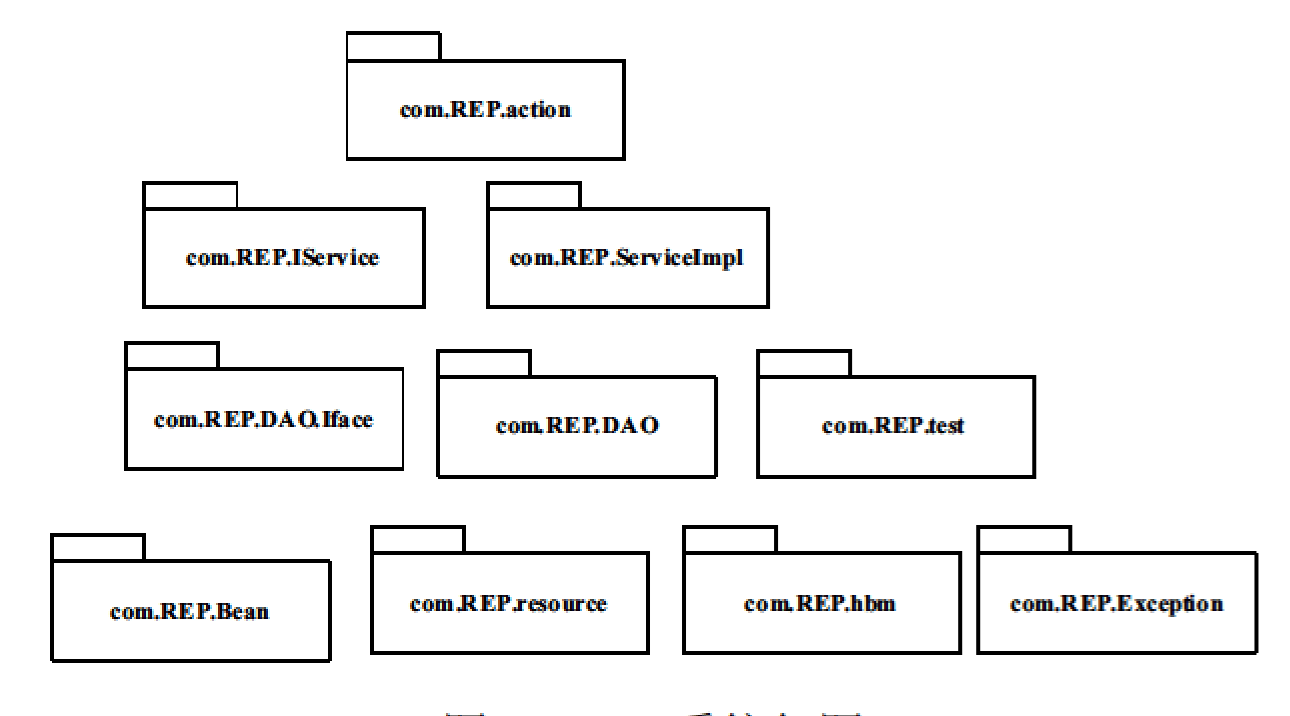


图 14-14　系统包图

其中：

- com.REP.action 包中主要包含了 Struts 中控制部分的类。
- com.REP.IService 存放各个业务逻辑层的接口类；com.REP.ServiceImpl 存放了各个业务逻辑实现的类。
- com.REP.DAO.Ifac 包含了进行数据访问需要实现的接口类；com.REP.DAO 包含了

数据访问的实现类；com.REP.test 包含了对 DAO 程序进行测试的程序。

- com.REP.Bean 包含了各个业务实体类；com.REP.resource 用于存放进行国际化所需要的资源文件；com.REP.hbm 用于存放各个业务实体对象所对应的对象-关系映射文件；com.REP.Exception 包含了各个系统可能发生的 Exception。

14.2.4　系统的 MVC 结构

本小节将介绍在餐费管理系统中 MVC 的各个部分内容。

- 视图部分如表 14-5 所示：

表 14-5　餐费管理系统的视图部分说明

组 成 部 分	文件名或类名	功　　能
	basePage.jsp	Tile 标签模板页面
	head.jsp	包含 banner 的页面
	menu.jsp	包含系统导航菜单的页面
	foot.jsp	包含版权信息的页面
	index.jsp	系统初始页面
	contentIndex.jsp	员工刷卡就餐页面，也是系统的具体初始页面
	repastSuccess.jsp	刷卡成功提示页面
	employeeRegist.jsp	员工注册页面
	banlancesSearch.jsp	就餐账户余额查询页面
	managerOperate.jsp	管理员登录页面
	fillAccount.jsp	管理员为员工账户充值页面
	modifyAccount.jsp	修改就餐账户页面
	viewAccount.jsp	显示就餐账户情况，对账户进行操作页面
	error.jsp	错误提示页面
	err404.jsp	404 错误提示页面
	err500.jsp	500 错误提示页面
ActionForm	org.apache.struts.action.ActionForm	动态 ActionForm，Struts 组件，用于保存表单数据

- 控制部分如表 14-6 所示：

表 14-6　餐费管理系统控制部分说明

组 成 部 分	文件名或类名	功　　能
Web 容器配置文件	Web.xml	配置 Servlet 等内容
Struts 配置文件	Struts-config.xml	对 Srtus 中的多种元素进行配置
Action	EmployeeOperateAction	和员工相关的控制程序
	EmployeeRegistAction	用于员工就餐账户注册
	ManagerOperateAction	和管理员相关的控制程序
	LogoutAction	退出系统

❑ 模型部分如表 14-7 所示：

表 14-7　餐费管理系统的模型部分说明

组 成 部 分	文件名或类名	功　　能
JavaBean	Employee	代表员工
	Account	代表就餐账户
	Manager	代表餐费管理员
业务逻辑	applicationContext.xml	Spring 配置文件用来装配业务逻辑组件
	IEmployeeService	和员工业务相关的接口
	IManagerService	和餐费管理员相关的业务接口
	EmployeeServiceImpl	实现和员工相关的业务
	ManagerServiceImpl	实现和管理员相关的业务
数据访问（DAO）	Hibernate.cfg.xml	Hibernate 配置文件，用于配置和数据库相关的信息
	Employee.hbm.xml	和员工对象相关的对象-关系映射文件
	Account.hbml.xml	和就餐账户对象相关的对象-关系映射文件
	Manager.hbm.xml	和管理员对象相关的对象-关系映射文件
	EmployeeDAO	与员工数据相关的数据访问程序
	AccountDAO	与账户数据相关的数据访问程序
	ManagerDAO	与管理员数据相关的数据访问程序

14.3　系统的开发环境

本章前面的几节内容介绍了系统的分析和设计等内容，本章后面几节将要把这些分析和设计变为现实。在介绍实现的内容时还按照 MVC 模式的组成，分别介绍了系统使用 Struts 框架来实现视图部分、控制部分，使用 Spring 框架来实现业务逻辑，使用 Hibenate 框架来实现数据访问。在 Eclipse 中对这 3 种框架都有很好的支持。系统的开发环境如下。

在 Windows XP 操作系统下进行 Eclipse 开发，具体版本是：

❑ Eclipse 3.1.0
❑ Struts 1.2.9
❑ Spring 1.2.8
❑ Hibernate 3.1.3
❑ Hibernate Synchronizer 3.1.9
❑ MySq l4.1.3+
❑ Tomcat 5.0.19

以上这些开发工具都是目前比较流行的开源软件，在这些开发工具中，Eclipse 在前面已经作过了介绍，这里不再说明。下面介绍其他几个开发工具的配置。建立一个新的 Tomcat 工程，然后就可以进行开发工具的配置。

14.3.1　Struts 在 Eclipse 中的配置

Struts 在 Eclipse 中的配置：

（1）将 Struts 压缩包中 lib 目录下的所有 Jar 包复制到 Web 应用项目的/WEB-INF/lib 目录下。

（2）在 Eclipse 环境下，在应用项目上右击，选择【properties】命令。

（3）进入【properties】窗口，单击【Java Build Path】选项。

（4）选择【libraries】选项卡，单击窗口右侧的【Add JARs】按钮，将项目 lib 中的所有 Jar 包引入。

如图 14-15 所示。

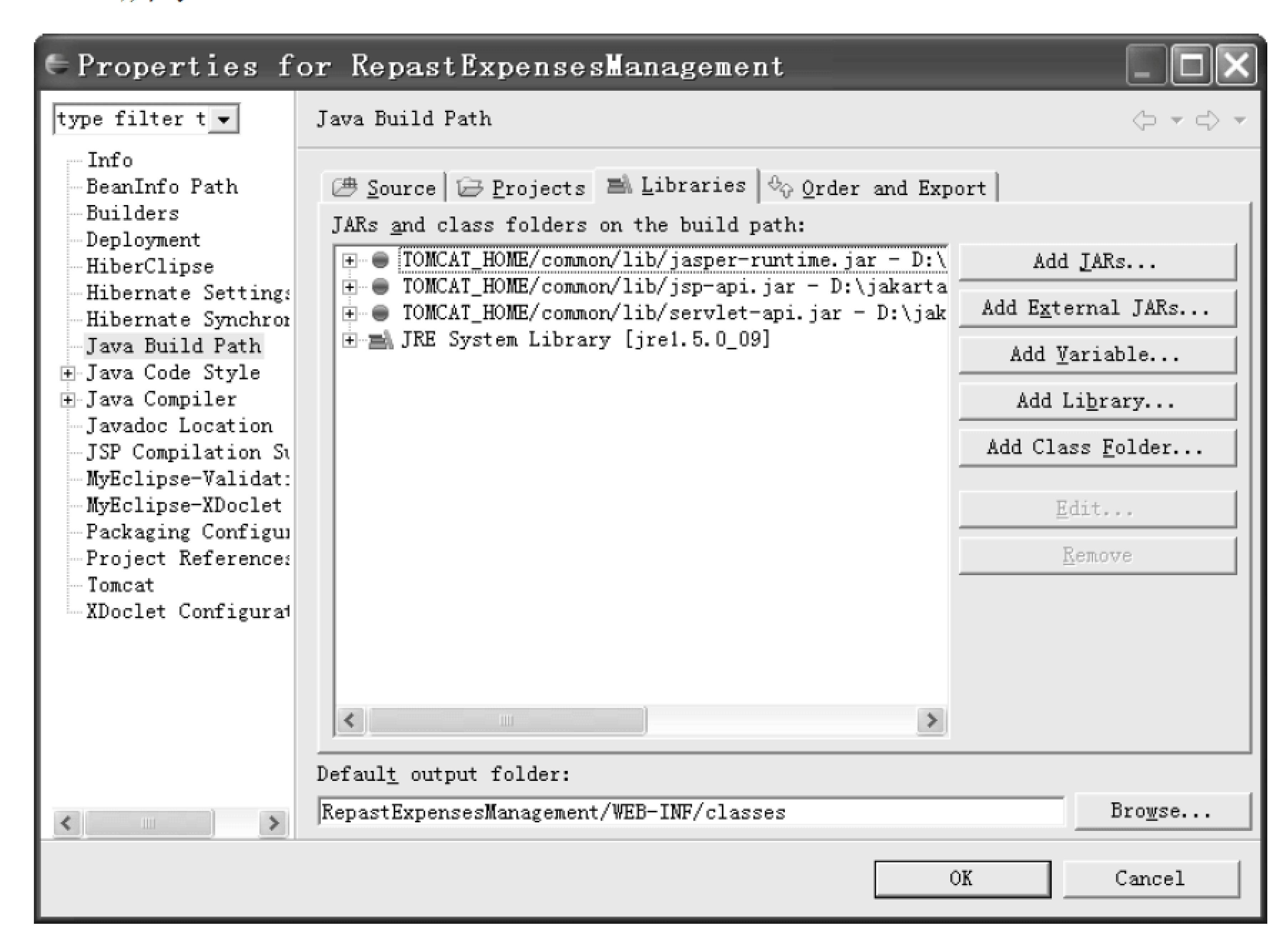


图 14-15　Eclipse 中手工引入 Struts Jar 包

要使得 Strus 框架能够使用，还需要在 Web 容器的配置文件 web.xml 中进行配置，代码如下所示：

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>

    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
```

```
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

14.3.2　Spring 在 Eclipse 中的配置

Spring 框架的配置方法基本和 Struts 相同，框架中的 Jar 包配置好后，也需要在 web.xml 文件中进行配置，代码如下所示：

```
<context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>context</servlet-name>
    <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
 </servlet>
```

经过配置后 Spring 的应用上下文就可以在 Web 容器启动后被加载。

14.3.3　Hibernate 在 Eclipse 中的配置

Hibernate 的配置和 Struts 和 Spring 的配置基本相同，所不同的是 Hibernate 不需要在 web.xml 文件中进行相应的配置。

14.3.4　Hibernate Synchronizer 在 Eclipse 中的配置

Hibernate Synchronizer 是一个数据驱动的代码自动生成工具，能够根据数据库中的内容生成相应的业务实体对象、数据访问对象（DAO），以及 Hibernate 配置文件 hibernate.cfg.xml 和对象-关系映射文件。利用这个工具能够很好地配合 Hibernate 框架下程序的开发，大大提高了开发效率。下面是 Hibernate Synchronizer 在 Eclipse 中的具体配置。

（1）将 HibernateSynchronizer-3.1.9.zip 解压缩，将解压后产生的 plugins 目录下的 com.hudson.hibernatesynchronizer_3.1.9 子目录复制到 Eclipse 下的 plugins 目录中，并重新启动 Eclipse。

（2）在新建工程上右击，选择【properties】，可以看到新打开的 properties 窗口左侧新增了一项“Hibernate Synchronizer”，如图 14-16 所示。

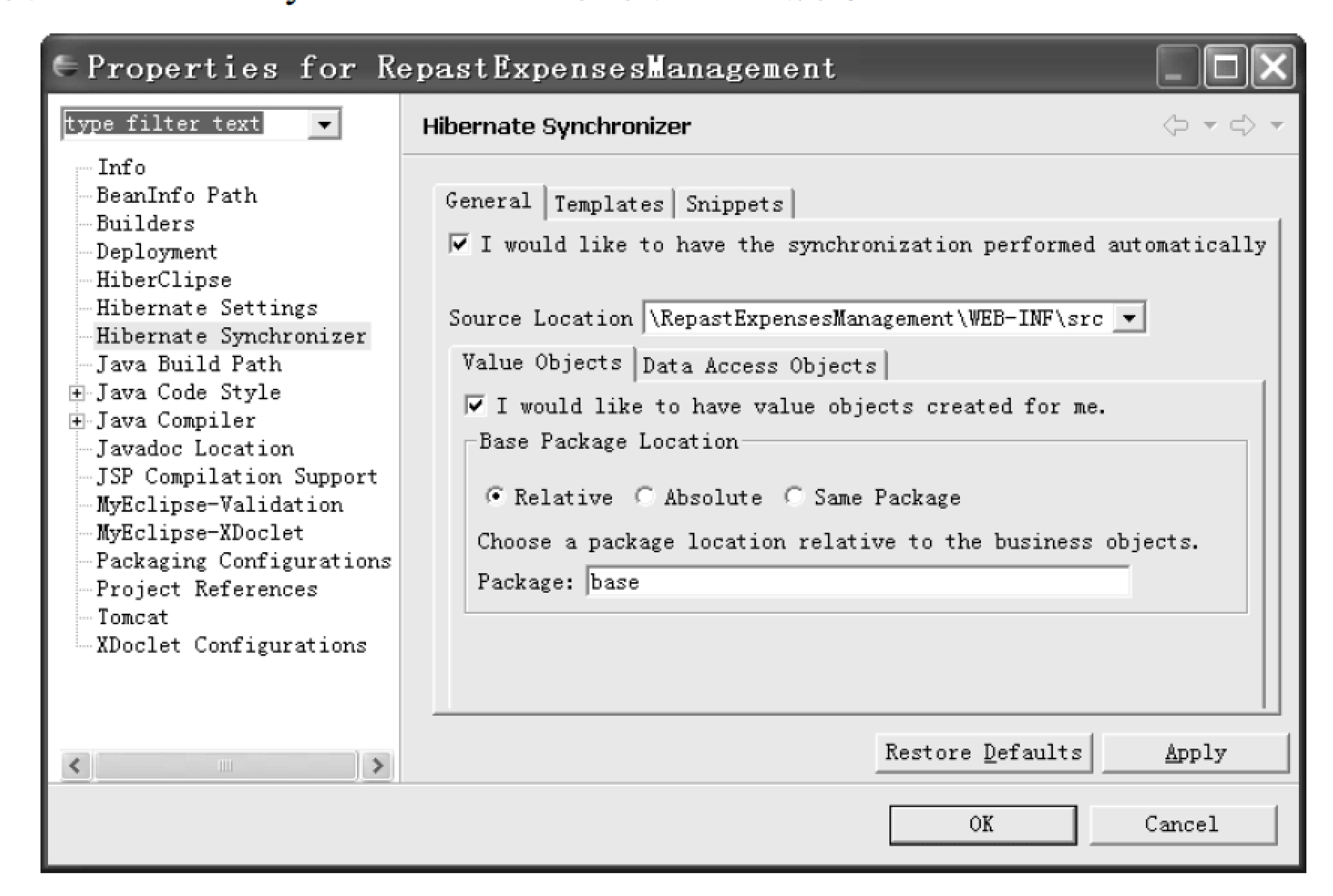


图 14-16　Hibernate Synchronizer 配置

（3）从图 14-16 中看到【properties】窗口的右侧可以进行 Hibernate Synchronizer 的相关配置，如业务实体对象的自动产生路径，DAO 对象的自动生成路径等。当然也可以选择默认的配置。

这样就可以在程序中使用 Hibernate Synchronizer 来根据数据库自动生成对象-关系映射文件、业务实体对象以及 DAO 对象了。MySql 和 Tomcat 的资料相对较多，前面章节也有涉及，读者可以参看相关的内容。

14.4　在 Eclipse 中用 Struts 建立视图

Strtus 是 Apache 软件研究基金下的 Jakarta 项目的子项目，是一个开源的框架。目前有很多项目采用了 MVC 模式，在实现方案上 Struts 是一个很好的选择。

在 MVC 模式中视图是其中的一个重要组成部分，在 MVC 模式中，视图负责显示从模型中采集的数据，也负责用户输入的数据和请求的传递，将这些数据和请求传递给控制器和模型。在 Struts 框架中，视图部分主要包括 JSP 页面和 ActionForm。

14.4.1　JSP 页面

JSP 页面用来显示模型中的数据、收集用户输入的数据以及提交用户的请求。另外在 Struts 框架中和 JSP 相关的组件还有 Struts 的标签库。下面是使用 Struts 标签的一个 JSP 页面片断：

```
<html:form action="employeeRegist.do" method="POST"   >
                    <br>
        <!-- 账户名称以及对应的文本框，其中账户名称名字符串从资源文件中获得-->
         <bean:message key="view.regist.repastCard"/>:
         <html:text property="repastCard" size="19"/><br>
         <!-- 如果账户名称文本框输入出现错误，则显示错误信息-->
         <logic:messagesPresent property="repastCard" >
                <html:messages id="repastCard1Msg" property="repastCard"/>
                           <font color="red">
               <bean:write name="repastCard1Msg"/>
               </font><br>
         </logic:messagesPresent>
          <!-- 确认账户名称以及对应的文本框-->
         <bean:message key="view.regist.repastCard2"/>:
         <html:text property="repastCard2" size="19"/><br>
         <logic:messagesPresent property="repastCard2" >
              <html:messages id="repastCard2Msg" property="repastCard2"/>
                         
              <font color="red">
                   <bean:write name="repastCard2Msg"/>
              </font><br>
         </logic:messagesPresent>
        …
         <BR><BR>
                  
         <input type="submit"   value="<bean:message key='view.regist'/>" />
        <input type="reset"     value="<bean:message key='view.reset'/>" />
</html:form>
```

从代码中可以看到页面中没有复杂的 Java 代码，只是一些和显示相关的内容，实现了 MVC 中显示和逻辑相分离的要求。

14.4.2　ActionForm

ActionForm 在 Struts 中专门用来传递表单数据，因此其方法中包含了对表单初始化的 reset()方法和验证表单的 validat()方法。ActionForm 也是 JavaBean，因此在 ActionForm 中也包含了 getXX()方法和 setXX()方法。ActionForm 在 Struts 中有两种：一种是静态 ActionForm；一种是动态 ActionForm。

在餐费管理系统中，Sturts 使用了 DynaActionForm，即动态 ActionForm。DynaActionForm 采用声明的方式，将表单的属性在配置文件中设定，如果需要改变某些属性只需改变配置文件即可，不存在代码重新编译的问题。代码如下所示：

```
<form-bean name="employeeRepastForm"type="org.apache.struts.validator.DynaValidatorForm" >
          <form-property name="repastCard" type="java.lang.String" />
          <form-property name="repastFee" type="java.lang.String" />
          <form-property name="repastCard2" type="java.lang.String" />
          <form-property name="employeeName" type="java.lang.String" />
```

```
        <form-property name="idCard" type="java.lang.String" />
    </form-bean>
```

14.5　在 Eclipse 中使用 Struts 建立 JSP 页面

在本节接下来的内容中将重点介绍如何在 Eclipse 中利用 Struts 框架来建立视图部分中的 JSP 页面。JSP 页面文件存在于 sshExample 项目下/pages 路径中。

14.5.1　建立模板页面

basePage.jsp 是 Tile 标签模板页面，是所有页面的基础。具体代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-tiles.tld"   prefix="tiles" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<html>
<head>
</head>
<body>
<div align="center">
<table border=0>
    <tr align="center">
        <td><!-- 包含了 logo 和 banner-->
      <tiles:insert attribute="header" />
        </td>
    </tr>
    <tr align="right">
        <td> <!-- 包含了系统菜单项-->
          <tiles:insert attribute="menu" />
        </td>
    </tr>
    <tr align="center">
        <td><!-- 包含了每个页面具体的内容-->
          <tiles:insert attribute="content" />
        </td>
    </tr>
    <tr align="center">
        <td><!-- 包含了 copyright 等内容-->
          <tiles:insert attribute="footer" />
        </td>
    </tr>
</table>
</div>
</body>
</html>
```

之所以要使用模板页面，是因为大量的页面中经常有重复的部分，如站点标记、系统导航菜单、页面版权信息等，为了避免重复地建立这些公共的内容，可以将这些内容分别放在不同的文件中，然后在每个页面中插入这些文件即可。而使用 Tile 标签建立模板页面将会使得这项工作变得非常简单。

14.5.2　建立 tiles-defs.xml

模板页面使用<tiles:insert>标记定义了页面每个组成部分，包括每个页面的不变部分和可变部分。这样可以在 tiles-defs.xml 文件中利用模板页面来建立一个复合页面，这个复合页面中各个不变部分被定义，留下可变部分没有定义，然后每个页面就可以扩展这个复合页面，只需定义其中的可变部分。tiles-defs.xml 文件的具体内容如下代码所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration//EN"        "d:/tilesdtd/tiles-config.dtd">

<tiles-definitions>
     <!--定义一个复合页面-->
  <definition   name="baseDefs" path="/pages/basePage.jsp" >
     <put name="header" value="header.jsp" />
     <put name="menu" value="menu.jsp" />
     <put name="content"   />
     <put name="footer" value="footer.jsp" />
   </definition>

     <!--定义应用的首页面-->
   <definition name="index" extends="baseDefs">
        <put name="content" value="ContentIndex.jsp" />
   </definition>
     <!--定义员工就餐刷卡页面-->
   <definition name="repastSuccess" extends="baseDefs">
        <put name="content" value="repastSuccess.jsp" />
   </definition>
     <!--定义员工注册就餐账户页面-->
   <definition name="employeeRegist" extends="baseDefs">
        <put name="content" value="employeeRegist.jsp" />
   </definition>
     <!--定义余额查询页面-->
     <definition name="banlancesSearch" extends="baseDefs">
        <put name="content" value="banlancesSearch.jsp" />
   </definition>
     <!--定义管理员查看员工账户以及操作页面-->
     <definition name="viewAccount" extends="baseDefs">
        <put name="content" value="viewAccount.jsp" />
   </definition>
     <!--定义管理员登录验证页面-->
     <definition name="managerOperate" extends="baseDefs">
```

```
            <put name="content" value="managerOperate.jsp" />
    </definition>
      <!--定义管理员修改账户页面-->
      <definition name="modifyAccount" extends="baseDefs">
            <put name="content" value="modifyAccount.jsp" />
    </definition>
      <!--定义账户充值页面-->
      <definition name="fillAccount" extends="baseDefs">
            <put name="content" value="fillAccount.jsp" />
    </definition>
      <!--定义错误提示页面-->
      <definition name="error" extends="baseDefs">
            <put name="content" value="/pages/error.jsp" />
    </definition>
</tiles-definitions>
```

可以看到配置文件中定义了一个复合页面，这个复合页面建立在模板页面的基础之上，在模板页面分别插入了页面的不变部分，如 head、menu、footer，还有页面中的可变部分，如 content。可以看到 content 并没有对应插入 JSP 文件。

但是在下面定义的页面中扩展了这个复合页面，并在其中填充了 content 的内容。这样新建立的页面只需建立页面中的可变部分，并在配置文件中插入到 content 对应的 value 中即可建立完整的页面。

当然，要使得在 tiles-defs.xml 定义的 JSP 页面在 Struts 中起作用，需要分别在 Web 容器的配置文件 web.xml 中和 Strus 的配置文件 struts-config.xml 中进行相关配置。在 web.xml 文件中的配置如下：

```
<taglib>
        <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
   </taglib>
```

这部分配置工作使得 Tiles 标签可以在 JSP 中使用。

在 struts-config.xml 文件中的配置如下：

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
   <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
</plug-in>
```

这部分配置使得 tiles-defs.xml 在 Struts 中可以起作用。如下代码表示了 tiles-defs.xml 在 Struts 的使用：

```
<action path="/logout"
                        scope="request"
                        validate="false"
                        input="error"
                        type="com.REP.action.LogoutAction">
                        <forward path="index" name="Index">
                        </forward>
           </action>
```

这是在 struts-config.xml 文件中定义的一个<action>元素，其中<forward>元素中的 path 属性值表示控制跳转的一个页面，path 值并不是一个实际的页面路径，而是 tiles-defs.xml 文件中定义的 index 页面。

14.6　在 Eclipse 中使用 Struts 建立页面的不变部分

下面将继续介绍页面中的不变部分，即 head、menu、footer 部分。

14.6.1　建立 Banner 页面

这个文件只有一行代码，用于显示一个图片，这个图片代表了信息反馈平台的站点 logo 和 banner：

```
<img src="<%=request.getContextPath()%>/pics/banner.jpg" width="800" height="70" border="0"
alt=""/>
```

14.6.2　建立菜单导航页面

这个文件引入了 6 个超链接，用来构成餐费管理系统的导航菜单：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<br>
<br>
<font size=2>
<a   href="index.do"><bean:message key="view.menu.mainPage"/>|</a>
<a href="goEmployeeRegist.do"> <bean:message key="view.menu.employeeRegist"/>|</a>
<a href="index.do"> <bean:message key="view.menu.employeeRepast"/>|</a>
<a href="goBanlancesSearch.do"> <bean:message key="view.menu.banlancesSearch"/>|</a>
<a href="goManagerOperate.do"> <bean:message key="view.menu.managerOperate"/>|</a>
<a href="logout.do"><bean:message key="view.menu.logout"/></a>
</font>
```

14.6.3　建立版权页面

这个页面可以包含一些帮助、版权等信息：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<br>
<br>
<br>
```

```
<bean:message  key="view.footer.aboutUs"/> | <bean:message  key="view.footer.connectUs"/> |
<bean:message key="view.footer.qa"/> | <bean:message key="view.footer.reference"/>
<P>                    • 2006 REP                    </P>
```

14.7　在 Eclipse 中使用 Struts 实现国际化

在前面介绍<bean:message>时，用到了资源文件，资源文件是国际化应用的基础。资源包其实就是由一系列具有相同前缀并且扩展名为.properties 的资源文件组成，如下列文件：

- ApplicationResources_zh.properties
- ApplicationResources_en.properties
- ApplicationResources.properties

上面这些资源文件具有相同的前缀 ApplicationResources，具有相同的扩展名.properties。这些文件一起组成了一个资源包。

这些文件中前缀后面的 en 和 zh 代表不同的语言编码，系统会根据应用环境去读取资源包中表示不同语言环境中的资源文件。资源文件中的内容一般以多个成对出现的 key 和 value 来组成，如下列代码：

```
error.password.length= password length is required at last {0}.
error.username.required=username is required.
error.password.required=password is required.
error.length={0} length is requird at last {1}.
```

等号左边的称为 key，等号右边的称为 value，这些内容可以包括需要国际化的文字、图片以及错误信息等。资源文件中可以包含参数，如上面代码中的“0”和“1”。在餐费管理系统中用到的资源文件如下：

```
#index.jsp 使用了下面的内容
view.project.name=餐费管理系统

#menu.jsp 使用了下面的内容
view.menu.mainPage=首页
view.menu.employeeRegist=注册
view.menu.employeeRepast=刷卡就餐
view.menu.banlancesSearch=查询余额
view.menu.managerOperate=管理员操作
view.menu.logout=退出

#footer.jsp 使用了下面的内容
view.footer.aboutUs=关于我们
view.footer.connectUs=联系我们
view.footer.qa=系统问答
view.footer.reference=使用手册

#ContentIndex.jsp 使用了下面的内容
view.title.repast=员工刷卡就餐
```

```
view.repastCard=请输入就餐卡号
view.repastFee=输入本次餐费标准
view.submit=刷卡
view.reset=重置

#repastSuccess.jsp 使用了下面的内容
view.viewrepastCard=您的卡号
view.viewrepastFee=本次消费(元)
view.repastSuccess=刷卡成功，谢谢!
view.account.banlances=账户余额
view.account.overDrawNumber=透支次数

#banlancesSearch.jsp 使用了下面的内容
view.title.searchBanlance=查询余额
view.search=查询

#employeeRegist.jsp  中用到的内容：
view.title.employeeRegist=员工注册
view.regist.repastCard=就餐卡号
view.regist.repastCard2=确认卡号
view.regist.idCard=身份证号
view.regist.employeeName=真实姓名
view.regist=注册

#managerOperate.jsp  中用到的内容：
view.title.managerLogin=管理员登录
view.managerLogin.loginName=登录名称
view.managerLogin.password=密码
view.login=登录

#viewAccount.jsp  中用到的内容：
view.account.managerOperate=用餐账号管理
view.account.repastCard=就餐账号
view.operate.modifyRepastCard=修改账号
view.operate.delRepastCard=删除账号
view.operate.fillAccount=账号充值
view.operate.modify=修改
view.operate.del=删除
view.operate.fillAccount=充值

#modifyAccount.jsp  中用到的内容：
view.title.modifyAccount=修改账号
view.old.repastCard=旧的账号
view.new.repastCard=请输入新的账号
view.modify=修改

#modifyAccount.jsp  中用到的内容：
view.fillAccount.banlance=请充值
view.title.fillAccount=账号充值
```

```
view.fill=充值

#EmployeeOperateAction.java 中用到的内容：
error.repast.accountNotExist=对不起，您的卡号不存在，您不能在此就餐
error.repast.OverDraw=对不起，您的账户已经透支过多，您不能在此就餐
error.regist.accountIsExist=对不起，账户已经存在，请选择其他卡号注册
error.repast.employeeNotExist=对不起，您输入的员工姓名和身份证号错误，注册失败！
error.regist.employeeHaveCard=员工已经注册过，不允许多次注册！

#ManagerOperateAction.java 中用到的内容：
error.loginName.required=用户名不能少
error.password.required=密码不能少
error.manangerlogin.managerNotExit=对不起，您输入的用户名或密码不正确或用户不存在！
error.account.accountIsExist=您输入的账号已经存在，请选择其他账号
error.account.accountNotExist=账号不存在

#下面是页面用到的通用错误提示
errors.required={0} 必须存在.
errors.minlength={0} 不能少于 {1} 个.
errors.maxlength={0} 不能大于 {1} 个.
errors.invalid={0}是无效的.
errors.byte={0} 必须是 byte 类型.
errors.short={0} 必须是 short 类型.
errors.integer={0} 必须是 integer 类型.
errors.long={0} 必须是 long 类型.
errors.float={0} 必须是 float 类型.
errors.double={0}必须是 double 类型.
errors.date={0} 不是一个 date 类型.
errors.range={0} 没有在{1}和{2}之间.
errors.creditcard={0} 是一个无效的.
errors.email={0} 是无效的.
errors.validwhen=两次输入内容不一致

#下面是 validation.xml 中用到的内容，用于表单验证
employeeRepastForm.repastCard=卡号
employeeRepastForm.repastCard2=确认的卡号
employeeRepastForm.employeeName=员工姓名
employeeRepastForm.idCard=员工身份证号
```

上面这个资源文件定义了餐费管理系统中的所有需要显示的内容，因此在页面中就可以使用<bean:message>标签来读取这些内容，而页面中不会出现具体的语言显示，从而实现了国际化。关于其他语言环境的资源文件在这里就不再一一列举了。

14.8　在 Eclipse 中使用 Struts 建立页面的可变部分

所谓页面的可变部分也就是前面介绍过的模板页面中 content 对应的页面。

14.8.1　员工就餐刷卡页面

contentIndex.jsp：员工就餐刷卡页面。这个页面主要是员工在就餐时使用，员工在就餐时需要输入就餐的卡号（也就是就餐账户的名称），还需要输入本次就餐的标准，比如 10 元或者 5 元等。输入完毕后提交。代码如下：

```
<%@ taglib uri="/WEB-INF/struts-logic.tld"   prefix="logic" %>
<html>
<head>
    <title><bean:message key="view.title.repast"/> </title>
</head>
<body>

            <!-- 用于显示员工刷卡后产生的错误信息-->
          <font color="red"> <html:errors name="repastErrors" />    </font>
          <html:form action="employeeOperateAction.do?method=employeeRepast" method=
"POST">
                        <br>
                        <!-- 员工刷卡使用的文本框，这里的刷卡实际上就是输入用户账号的名称-->
                          <bean:message key="view.repastCard"/>：<br>
                          <html:text property="repastCard" size="19"/><br><br>
                          <bean:message key="view.repastFee"/>(RMB)：<br>
                          <html:text property="repastFee" size="19"/><br>
                          <BR><BR>
                        <input type="submit"   value="<bean:message key='view.submit'/>"/>
                        <input type="reset"     value="<bean:message key='view.reset'/>" />
            </html:form>

</body>
</html>
```

14.8.2　员工刷卡成功页面

repastSuccess.jsp：员工就餐刷卡后显示出相应的信息。如果员工刷卡后系统没有提示错误信息，如账户不存在或者账户透支次数超过 3 次等，就说明刷卡成功，否则刷卡失败。刷卡成功后会显示出本页面的内容，即成功提示和账户余额。代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"   prefix="logic" %>
<html>
<head>
    <title><bean:message key="view.title.repast"/> </title>
</head>
<body>
```

```
        <html:form action="employeeOperateAction.do" method="POST">
                <br>
                <!-- 员工刷卡成功后显示出余额和透支次数等内容-->
                       
                 <!-- 员工刷卡成功后显示就餐卡号-->
                 <bean:message key="view.viewrepastCard"/>:
                 <html:text property="repastCard" size="19"/><br>
                 <!-- 员工刷卡成功后显示出消费情况-->
                 <bean:message key="view.viewrepastFee"/>:
                 <html:text property="repastFee" size="19"/><br><br>
                 <!-- 员工刷卡成功后显示刷卡成功信息-->
                 <bean:message key="view.repastSuccess"/><br><br>
                  <!-- 员工刷卡成功后显示账户余额 -->
                 <bean:message key="view.account.banlances"/>:
                    <bean:write name="banlances"/><br>
                    <!-- 员工刷卡成功后显示账户透支次数 -->
                 <bean:message key="view.account.overDrawNumber"/>:
                 <bean:write name="overDrawNub"/>
        </html:form>
</body>
</html>
```

14.8.3　员工账户注册页面

employeeRegist.jsp：员工进行账户注册。员工在能够进行刷卡就餐前需要申请一个账户，这就需要使用本页面进行注册，注册时需要在注册表单中填写账户名称、账户名称确认、员工姓名、员工身份证号。这里输入员工身份证号和姓名主要用来判断这个员工是否存在。代码如下：

```
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head><title><bean:message key="view.title.employeeRegist"/></title></head>
<body>
<font color="red"> <html:errors name="registErrors" />    </font>

<html:form action="employeeRegist.do" method="POST"
                        onsubmit="return validateEmployeeRepastForm(this);">
                <br>
                <!-- 账户名称以及对应的文本框，其中账户名称名字符串从资源文件中获得-->
                 <bean:message key="view.regist.repastCard"/>:
                 <html:text property="repastCard" size="19"/><br>
                 <!-- 如果账户名称文本框输入出现错误，则显示错误信息-->
                 <logic:messagesPresent property="repastCard" >
```

```
            <html:messages id="repastCard1Msg" property="repastCard"/>
                       
           <font color="red"><bean:write name="repastCard1Msg"/></font><br>
        </logic:messagesPresent>

         <!-- 确认账户名称以及对应的文本框-->
        <bean:message key="view.regist.repastCard2"/>:
         <html:text property="repastCard2" size="19"/><br>
         <!-- 如果确认账户名称时文本框输入出现错误，则显示错误信息-->
        <logic:messagesPresent property="repastCard2" >
            <html:messages id="repastCard2Msg" property="repastCard2"/>
                        
              <font color="red"><bean:write name="repastCard2Msg"/></font><br>
        </logic:messagesPresent>

        <!-- 员工姓名及对应的文本框-->
        <bean:message key="view.regist.employeeName"/>:
         <html:text property="employeeName" size="19"/><br>
        <!-- 如果员工姓名文本框输入时出现错误，则显示错误信息-->
        <logic:messagesPresent property="employeeName" >
             <html:messages id="employeeNameMsg" property="employeeName"/>
                         
             <font color="red"><bean:write name="employeeNameMsg"/></font>
        </logic:messagesPresent>

        <!-- 身份证号以及对应的文本框-->
        <bean:message key="view.regist.idCard"/>:
         <html:text property="idCard" size="19"/><br>
        <!-- 身份证号文本框输入时出现错误，则显示错误信息-->
        <logic:messagesPresent property="idCard" >
            <html:messages id="idCardMsg" property="idCard"/>
                        
            <font color="red"><bean:write name="idCardMsg"/></font>
        </logic:messagesPresent>

        <BR><BR>
                 
        
        <input type="submit"   value="<bean:message key='view.regist'/>" />
       <input type="reset"     value="<bean:message key='view.reset'/>" />
<html:javascript   formName="employeeRepastForm" staticJavascript="false" />
</html:form>
</body>
</html>
```

在这个页面中多处用到了<logic: messagesPresent>标签，这个标签属于 Struts 的 Logic 标签库中的一个，Struts 的 Logic 标签库主要用来进行逻辑判断、循环迭代、流程的控制。<logic:messagePresent>标签用来判断指定的 ActionMessages 对象是否存在，以及对象中具体的消息是否存在。

14.8.4　员工账户查询页面

banlanceSearch.jsp：用于员工查询自己的账户余额和透支次数。如果员工账户透支次数超过 3 次则不允许就餐，因此在这个页面中可以查询员工的账户余额和透支次数，通过这个查询使用户知道自己的账户使用情况。代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"   prefix="logic" %>
<html>
<head>
    <title><bean:message key="view.title.searchBanlance"/> </title>
</head>
<body>

            <!-- 用于显示员工刷卡后产生的错误信息-->
          <font color="red"> <html:errors name="repastErrors" />    </font>
          <html:form action="employeeOperateAction.do?method=banlancesSearch" method=
"POST">
                        <br>
                        <!-- 员工刷卡使用的文本框，这里的刷卡实际上就是输入用户账号的名称-->
                          <bean:message key="view.repastCard"/>：<br>
                          <html:text property="repastCard" size="19"/><br>
                          <BR><BR>
                          <!-- 如果 request 或 session 中存在余额，则显示出来-->
                          <logic:present name="banlances">
                              <bean:message key="view.account.banlances"/>：
                              <bean:write name="banlances"/><br>
                          </logic:present>
                          <!-- 如果 request 或 session 中存在透支次数，则显示出来-->
                            <logic:present name="overDrawNub">
                              <bean:message key="view.account.overDrawNumber"/>：
                              <bean:write name="overDrawNub"/><br>
                          </logic:present>
                          <br><br>
                         <input type="submit"   value="<bean:message key='view.search'/>"/>
                         <input type="reset"    value="<bean:message key='view.reset'/>" />
             </html:form>

</body>
</html>
```

14.8.5　管理员登录页面

managerOperate.jsp：用于管理员登录。管理员可进行员工账户的相关操作比如修改、

删除等。对于管理员要求进行身份的确认，本页面就是用来确认管理员身份的页面。在登录验证中，管理员要输入自己的用户名和密码。

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<html>
<head>
  <title><bean:message key="view.title.managerLogin"/></title>
</head>
<body>
          <br><br>
           <font color="red"> <html:errors   />    </font>
          <html:form action="managerOperateAction.do?method=managerLogin" method="post">
                    <br>
                    <!-- 登录名以及对应的输入文本框
                      <bean:message key="view.managerLogin.loginName"/>:
                      <html:text property="loginName" size="19"/><br>
                      <!--如果登录名输入出现错误将显示错误信息-->
                      <logic:messagesPresent property="loginName" >
                          <html:messages id="userNameMsg" property="loginName"/>
                                     
                            <font color="red"><bean:write name="userNameMsg"/></font><br>
                      </logic:messagesPresent>

                             
                      <!-- 登录密码以及对应的输入文本框-->
                      <bean:message key="view.managerLogin.password"/>:
                      <html:password property="password" size="20"/><br>
                      <!--如果登录密码输入出现错误将显示错误信息-->
                      <logic:messagesPresent property="password" >
                          <html:messages id="passwordMsg" property="password"/>
                                     
                            <font color="red"><bean:write name="passwordMsg"/></font>
                      </logic:messagesPresent>
                      <BR><BR>
                               
                      <input type="submit"   value="<bean:message key='view.login'/>"/>
                     <input type="reset"   value="<bean:message key='view.reset'/>"/>
          </html:form>
     </body>
</html>
```

14.8.6　管理员管理账户页面

viewAccount.jsp：用于管理员查看员工的账户，并对其进行操作。管理员在这个页面中将会看到员工账户的余额、透支次数，同时在本页面中还会显示几个操作按钮，如修改账

户、删除账户、账户充值，管理员单击相应的按钮后，会转到不同的操作页面。代码如下：

```
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld" prefix="logic" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %>
<%@page contentType="text/html;charset=UTF-8" language="java"%>
<title><bean:message key="view.account.managerOperate"/></title>

<h1 align="center"><bean:message key="view.account.managerOperate"/></h1>
<TABLE align="center" border="1" width="50%">
      <TR>
            <TD width="20%"><h3><bean:message key="view.account.repastCard"/></h3></TD>
            <TD width="20%"><h3><bean:message key="view.account.banlances"/></h3></TD>
            <TD width="15%"><h3><bean:message key="view.account.overDrawNumber"/></h3>
</TD>
            <TD width="15%"><h3><bean:message key="view.operate.modifyRepastCard"/></h3>
</TD>
            <TD width="15%"><h3><bean:message key="view.operate.delRepastCard"/></h3>
</TD>
           <TD width="15%"><h3><bean:message key="view.operate.fillAccount"/></h3></TD>
      </TR>
      <!-- 循环显示账户列表 accountList 中的内容-->
      <logic:iterate id="account" name="accountList">
      <TR>
            <!-- 显示账户的卡号，即账户的名称-->
            <TD><bean:write name="account" property="loginName"/></TD>
            <!-- 显示账户的余额-->
            <TD><bean:write name="account" property="balance"/></TD>
            <!-- 显示账户的透支次数-->
            <TD><bean:write name="account" property="overdrawNumber"/></TD>
            <!-- 对账户进行修改-->
            <TD><a    href="managerOperateAction.do?method=GoModifyAccount&id=<bean:write
name='account' property='id'/>"><bean:message key="view.operate.modify"/></a></TD>
           <!-- 对账户进行删除 -->
           <TD><a href="managerOperateAction.do?method=delAccount&id=<bean:write name=
'account' property='id'/>"><bean:message key="view.operate.del"/></a></TD>
           <!-- 对账户进行充值-->
           <TD><a href="managerOperateAction.do?method=GoFillAccount&id=<bean:write name=
'account' property='id'/>"><bean:message key="view.operate.fillAccount"/></a></TD>
      </TR>

      </logic:iterate>
</TABLE>
```

代码中<logic:iterate>用于对 name 属性值表示的变量进行循环迭代，<bean:write>标签用于显示 name 属性值对应的变量的属性值。

14.8.7　修改员工账户页面

modifyAccount.jsp 文件用于修改员工就餐账户的账号。当员工想要修改账户名称时使用本页面，在修改时需要输入员工原来账户的名称。代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"   prefix="logic" %>
<html>
<head>
    <title><bean:message key="view.title.modifyAccount"/> </title>
</head>
<body>

            <!-- 显示页面接受到的错误信息-->
          <font color="red"> <html:errors />     </font>
          <html:form action="managerOperateAction.do?method=modifyAccount"method= "POST">
                          <br>
                          <!-- 员工使用的旧的账号-->
                          <bean:message key="view.old.repastCard"/>：
                          <bean:write name="oldRepastCard"/><br>
                          <BR><BR>
                          <!-- 员工输入新的账号-->
                          <bean:message key="view.new.repastCard"/>：<br>
                          <html:text   property="repastCard"/><br>
                          <!-- 传递被操作的员工账号 id-->
                          <input type="hidden" name="id" value="<bean:write name='id'/>">
                          <br><br>
                 <input type="submit"value="<bean:message key='view.modify'/>"/>
                  <input type="reset"    value="<bean:message key='view.reset'/>" />
            </html:form>

</body>
</html>
```

14.8.8　员工账户充值页面

fillAccount.jsp 文件用于管理员对员工账户进行充值。如果员工账户余额不足，则应该及时充值。充值工作由管理员来完成，在充值时需要输入充入账户的数值。代码如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<%@ taglib uri="/WEB-INF/struts-logic.tld"   prefix="logic" %>
<html>
<head>
    <title><bean:message key="view.title.fillAccount"/> </title>
</head>
<body>

          <!-- 用于显示错误信息-->
         <font color="red"> <html:errors />     </font>
         <html:form action="managerOperateAction.do?method=fillAccount" method="POST">
                    <br>


                      <BR><BR>
                       <!-- 输入充值金额-->
                      <bean:message key="view.fillAccount.banlance"/>：<br>
                      <html:text   property="banlance"/><br>
                      <input type="hidden" name="id" value="<bean:write name='id'/>">
                      <br><br>
                     <input type="submit"    value="<bean:message key='view.fill'/>"/>
                     <input type="reset"     value="<bean:message key='view.reset'/>" />
          </html:form>

</body>
</html>
```

14.9　在 Eclipse 中用 Struts 建立控制部分

控制部分主要包括配置文件 web.xml、Struts-config.xml，还有自定义的 Action。

14.9.1　配置 web.xml

web.xml 位于 sshExample 项目下的/WEB-INF/路径下，web.xml 在前面介绍 Sruts 的配置时已经用到，这里将进一步介绍。

Web 服务器启动后将从 web.xml 文件中读取配置信息，并根据 web.xml 来装配 Web 组件，实现 Web 应用。同样，Struts 应用和 Spring 应用的初始加载也要依靠 web.xml，因此也需要在 web.xml 文件中来对 Struts 和 Spring 等内容进行配置。下面是餐费管理系统中 web.xml 的具体配置：

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/ XMLSchema
-instance"
version="2.4" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <!--
 <listener>
  <listener-class>
   org.springframework.web.context.ContextLoaderListener
  </listener-class>
 </listener>
  -->
<!-- 下面是 Spring 应用配置-->
 <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/applicationContext.xml</param-value>
  </context-param>
  <servlet>
    <servlet-name>context</servlet-name>
    <servlet-class>org.springframework.web.context.ContextLoaderServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
 </servlet>
 <!-- 下面是 Struts 应用配置-->
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>

    <init-param>
      <param-name>debug</param-name>
      <param-value>3</param-value>
    </init-param>
    <init-param>
      <param-name>detail</param-name>
      <param-value>3</param-value>
    </init-param>
    <load-on-startup>0</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
  <!-- 系统欢迎界面的配置-->
```

```
  <welcome-file-list>
        <welcome-file>/pages/index.jsp</welcome-file>
  </welcome-file-list>

   <!--  下面开始定义错误处理页面-->
   <!--
  <error-page>

     <error-code>404</error-code>
     <location>/pages/err404.jsp</location>

  </error-page>

  <error-page>
     <error-code>500</error-code>
     <location>/pages/err500.jsp</location>
  </error-page>
  <error-page>
      <exception-type>javax.servlet.ServletException</exception-type>
      <location>/pages/errServletException.jsp</location>
  </error-page>
  -->
  <!--错误处理页面定义结束-->

 <jsp-config>
 <!--  下面开始引入 Struts 标记库-->
  <taglib>
      <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
  </taglib>
  <taglib>
      <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
  </taglib>
  <taglib>
      <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
  </taglib>
  <taglib>
      <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
      <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
  </taglib>
  <!--Struts 标记引入结束-->
  </jsp-config>

</web-app>
```

14.9.2　配置 struts-config.xml

struts-config.xml 位于 sshExample 项目下的/WEB-INF/路径下，Strtus-config.xml 是 Struts 框架中一个重要的配置文件，通过 struts-config.xml 可以完成对 ActoionForm 的定义、用户请求和 Action 之间的映射、页面的跳转，以及其他较为重要的配置。在 struts-config.xml 文件的配置中，<action-mapping>元素的配置是一个重要的内容。

当用户提交表单或进行转发动作时，会发出扩展名为.do 的请求，根据 web.xml 的配置，ActionServlet 会接受*.do 的请求，并且会查找 struts-config.xml 文件的<action-mappings>元素，寻找可以处理这种请求的<action>元素是否存在，如果存在则进行相应的处理。

<action-mappings>是 strut-config.xml 中比较重要的元素，利用<action-mappings>元素可以帮助系统实现流程的控制，<action-mappings>元素将用户的请求和对请求的处理联系起来。其具体语法结构如下：

```
<action-mappings >
        <action path="访问 Action 的路径 "
                  name="和 Action 对应的 formbean 标示"
                  scope="ActionForm Bean 的使用范围"
                  input="错误处理页面路径"
                  type="请求处理类的完整路径"
                  validate="是否使用 formbean 的验证方法"
            <forward path="转发路径 " name="转发标示">
            <!--forward 元素可以有多个-->
        >
        </action>
        <!--action 元素可以有多个-->
</action-mappings>
```

其中，<action>元素的 path 属性值表示了对客户发出的具体请求，type 属性值表示对请求进行处理的具体程序，这个程序可以是开发者自定义的 Action，如 com.REP.action 下面定义的 Action，也可以是 Struts 预定义的 Action，如 org.apache.struts.actions.ForwardAction。

下面是餐费管理系统中具体的 struts-config.xml 文件的配置：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
1.2//EN" "http://struts.apache.org/dtds/struts-config_1_2.dtd">
<!--struts-config.xml for login example-->
<struts-config>
<!--form-beans 用来定义 ActionForm-->
  <form-beans >
   <!--配置和员工操作相关的 ActionForm-->
   <form-bean name="employeeRepastForm" type="org.apache.struts.validator.DynaValidator
Form" >
          <form-property name="repastCard" type="java.lang.String" />
          <form-property name="repastFee" type="java.lang.String" />
          <form-property name="repastCard2" type="java.lang.String" />
```

```
            <form-property name="employeeName" type="java.lang.String" />
            <form-property name="idCard" type="java.lang.String" />
    </form-bean>
        <!--配置和管理员操作相关的 ActionForm-->
    <form-beanname="managerOperateForm" type="org.apache.struts.validator.DynaValidatorForm" >
            <form-property name="repastCard" type="java.lang.String" />
            <form-property name="loginName" type="java.lang.String" />
            <form-property name="password" type="java.lang.String" />
            <form-property name="banlance" type="java.lang.String" />
            <form-property name="overdrawNumber" type="java.lang.String" />
    </form-bean>
  </form-beans>
 <!--global-forwards  定义全局异常处理-->
  <global-exceptions>
        <exception key="error.db.getSave"
                                path="/pages/error.jsp"
                                scope="request"
                                type="java.sql.SQLException" />
  </global-exceptions>
<!--global-forwards  定义全局转发关系-->
  <global-forwards >
        <forward name="welcome" redirect="true" path="/pages/welcome.jsp"/>
        <forward name="error"     path="error"/>
       <!--下面可以定义多个 forward 元素-->
  </global-forwards>

<!--action-mappings 用来建立用户请求和 Action 的映射-->
  <action-mappings >

        <!--配置处理就餐刷卡请求的 Action-->
        <action path="/index"
                scope="request"
                validate="false"
                type="org.apache.struts.actions.ForwardAction"
                parameter="index"/>
        <!--配置跳转到余额查询请求的 Action-->
        <action path="/goBanlancesSearch"
                scope="request"
                validate="false"
                type="org.apache.struts.actions.ForwardAction"
                parameter="banlancesSearch"/>
        <!--配置跳转到员工注册请求的 Action-->
        <action path="/goEmployeeRegist"
                scope="request"
                validate="false"
                type="org.apache.struts.actions.ForwardAction"
                parameter="employeeRegist"/>
        <!--配置跳转到管理员登录请求的 Action-->
```

```
        <action path="/goManagerOperate"
                scope="request"
                validate="false"
                type="org.apache.struts.actions.ForwardAction"
                parameter="managerOperate"/>
        <!--配置系统退出请求的 Action-->
        <action path="/logout"
                scope="request"
                validate="false"
                input="error"
                type="com.REP.action.LogoutAction">
                <forward path="index" name="Index">
                </forward>
        </action>
        <!--配置处理员工注册请求的 Action-->
        <action path="/employeeRegist"
                scope="request"
                validate="true"
                input="error"
                name="employeeRepastForm"
                type="com.REP.action.EmployeeRegistAction"
                parameter="method">
                <forward path="index" name="Index"/>
          </action>

        <!--配置处理员工操作请求的 Action-->
        <action path="/employeeOperateAction"
                scope="request"
                validate="false"
                input="error"
                name="employeeRepastForm"
                type="com.REP.action.EmployeeOperateAction"
                parameter="method">
                <forward path="repastSuccess" name="RepastSuccess"/>
                <forward path="employeeRegist" name="EmployeeRegist"/>
                <forward path="index" name="Index"/>
                <forward path="banlancesSearch" name="BanlancesSearch"/>
          </action>
        <!--配置处理管理员操作请求的 Action-->
          <action path="/managerOperateAction"
                scope="request"
                validate="false"
                input="error"
                name="managerOperateForm"
                type="com.REP.action.ManagerOperateAction"
                parameter="method">
                <forward path="managerOperate" name="ManagerLogin"/>
                 <forward path="/managerOperateAction.do?method=viewAccount" name="Go
ViewAccount"/>
```

```
                    <forward path="viewAccount" name="ViewAccount"/>
                    <forward path="modifyAccount" name="ModifyAccount"/>
                    <forward path="fillAccount" name="FillAccount"/>
            </action>
  </action-mappings>
  <!--自定义 RequestProcessor，用于解决汉字乱码问题-->
  <controller processorClass="com.REP.processor.EncodingProcessor"/>
  <!-- 配置资源文件-->
  <message-resources   parameter="com.REP.resource.ApplicationResources" />
  <!-- 配置 Tile 插件-->
 <plug-in className="org.apache.struts.tiles.TilesPlugin">
  <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
 </plug-in>
     <!-- 配置 validator 验证插件-->
 <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.
xml" />
  <set-property property="stopOnFirstError" value="false" ></set-property>
 </plug-in>
     <!-- 配置 Spring 插件-->
 <plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
        <set-property property="contextConfigLocation" value="/WEB-INF/applicationContext.xml">
          </set-property>
  </plug-in>
</struts-config>
```

需要注意的是，struts-config.xml 文件中的各个元素之间的先后顺序不能改变。

14.9.3　建立 Action

在 Action 的定义中，一部分使用了 Struts 的预定义 Action，如 ForwardAction，另外一部分属于自定义的 Action。在系统中为了按照 MVC 模型的设计思想，页面之间的跳转没有直接写在页面中进行跳转，而是交给了控制部分来处理，具体方法就是在<action>元素中将 type 属性值设为 ForwardAction，这样页面跳转的请求被送到 ActionServlet 来处理，最终实现将请求转发到相应的页面。

ForwardAction 属于 Struts 预定义，这里就不再列举其代码内容，读者可以查阅相关资料。另外，由于自定义的 Action 内容较多，就放在下一节单独介绍。

14.10　自定义的 Action

自定义的 Action 位于 RepastExpensesManagerment 项目下 com.REP.action 包中。

14.10.1　处理员工注册请求的 Action

EmployeeRegistAction：用于处理员工注册请求。在前面的员工注册页面中会发出这样的请求：

```
<html:form action="employeeRegist.do" method="post">
```

根据这个请求将会建立下面的 Action 进行处理。

```
public class EmployeeRegistAction extends Action {
    private IEmployeeService employeeservice;
    public ActionForward execute(ActionMapping mapping,
            ActionForm form, HttpServletRequest request,
            HttpServletResponse response)   {

        //从页面中获得员工输入的信息
        DynaActionForm dform = (DynaActionForm)form;
        //获得员工输入的就餐卡号
        String repastCard = dform.getString("repastCard");
        //获得员工输入的身份证号
        String idCard = dform.getString("idCard");
        ////获得员工输入的真实姓名
        String employeeName = dform.getString("employeeName");

         try {
            //调用员工相关业务逻辑，实现员工注册
            employeeservice.registByName(repastCard,employeeName,idCard);
         } catch (EmployeeBeUsedException e) {
            //建立 ActionMessages 对象
            ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,
                                  new ActionMessage("error.regist.employeeHaveCard"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
         } catch (AccountIsExistException e) {
         //建立 ActionMessages 对象
         ActionMessages errors = new ActionMessages();
         //将异常或错误信息存入 ActionMessages 对象 errors 中
         errors.add(ActionMessages.GLOBAL_MESSAGE,
                                  new ActionMessage("error.regist.accountIsExist"));
         //把 ActionMessages 对象存入到 request 对象中
         saveErrors(request,errors);
         //跳转到错误处理页面
         return mapping.getInputForward();
        } catch (EmployeeNotExistException e) {
```

```
                //建立 ActionMessages 对象
                ActionMessages errors = new ActionMessages();
                //将异常或错误信息存入 ActionMessages 对象 errors 中
                errors.add(ActionMessages.GLOBAL_MESSAGE,
                                        new ActionMessage("error.regist.employeeNotExist"));
                //把 ActionMessages 对象存入到 request 对象中
                saveErrors(request,errors);
                //跳转到错误处理页面
                return mapping.getInputForward();
            }

            return mapping.findForward("Index");


        }
        public IEmployeeService getEmployeeservice() {
            return employeeservice;
        }
        public void setEmployeeservice(IEmployeeService employeeservice) {
            this.employeeservice = employeeservice;
        }
}
```

Action 中首先从页面获得员工输入的信息，然后根据这些信息调用员工相关业务逻辑 employeeservice 的 registByName()方法，实现了注册，注册过程中会产生一些 Exception，Action 中将对这些 Exception 进行处理，并跳转到相应的错误处理页面，如果注册成功，将会跳转到员工就餐刷卡页面。

在程序中自定义的 Action 继承自 org.apache.struts.action.Action，在这种 Action 中必须要重写 execute()方法。在 Action 中使用的员工业务逻辑 employeeservice 的初始化通过 setXXX()方法来完成，这样就可以利用 Spring 在运行期加载所需要的员工业务逻辑。

14.10.2　处理员工其他请求的 Action

EmployeeOperateAction 用于处理员工某些操作的请求。员工可能发出这样的请求：

```
<html:form action="employeeOperateAction.do?method=employeeRepast" method="POST">
```

这个请求中包含了参数 method，对于类似这样的请求，可以使用下面的 DispatchAction 进行处理：

```
public class EmployeeOperateAction extends DispatchAction {

    IEmployeeService employeeservice;
    //处理员工就餐刷卡请求
     public ActionForward employeeRepast(ActionMapping mapping,
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response)   {
```

```
        /*
         *如果需要对员工刷卡的位置有具体的要求，比如只允许员工在餐厅的某台计算机上刷卡就餐，
         *就可以利用 request.getRemoteHost()方法获得用户使用计算机的 ip 地址，如果是正
         *确的 ip 地址则允许刷卡，否则禁止刷卡。读者可以自己添加这部分内容。
         */

        //获得员工页面输入内容，主要是就餐账户名称、消费金额
        DynaActionForm dform = (DynaActionForm)form;
        String repastCard =dform.getString("repastCard");
        String repastFee =dform.getString("repastFee");

        try {
            //对输入内容进行处理，完成刷卡动作
            employeeservice.repast(repastCard,repastFee);
            //获得账户余额
            String banlances =String.valueOf(employeeservice.searchBanlances(repastCard));
            //获得透支次数
            String overDrawNub = String.valueOf(employeeservice.searchOverDrawNub(repast
Card));

            //将账户余额和透支次数存放在 Request 范围内，方便其他页面调用
            request.setAttribute("banlances",banlances);
            request.setAttribute("overDrawNub",overDrawNub);
            return mapping.findForward("RepastSuccess");

        } catch (AccountNotExistException e) {
            //建立 ActionMessages 对象
            ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,
                        new ActionMessage("error.repast.accountNotExist"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
        } catch (OverDrawException e) {
            //建立 ActionMessages 对象
            ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,new  ActionMessage("error.repast.
OverDraw"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
        }
    }
    //处理员工余额查询请求
    public ActionForward banlancesSearch(ActionMapping mapping,
```

```
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response)   {

            //从页面获得员工输入的就餐卡号
            DynaActionForm dform = (DynaActionForm)form;
            String repastCard =dform.getString("repastCard");

            String banlances;
            try {
                //查询余额
                banlances = String.valueOf(employeeservice.searchBanlances(repastCard));
                String overDrawNub = String.valueOf(employeeservice.searchOverDrawNub(repast
Card));
                request.setAttribute("banlances",banlances);
                request.setAttribute("overDrawNub",overDrawNub);
            } catch (AccountNotExistException e) {
                //建立 ActionMessages 对象
                ActionMessages errors = new ActionMessages();
                //将异常或错误信息存入 ActionMessages 对象 errors 中
                errors.add(ActionMessages.GLOBAL_MESSAGE,
                            new ActionMessage("error.repast.accountNotExist"));
                //把 ActionMessages 对象存入到 request 对象中
                saveErrors(request,errors);
                //跳转到错误处理页面
                return mapping.getInputForward();
            }

            return mapping.findForward("BanlancesSearch");
        }
        public IEmployeeService getEmployeeservice() {
            return employeeservice;
        }
        public void setEmployeeservice(IEmployeeService employeeservice) {
            this.employeeservice = employeeservice;
        }
}
```

从代码中可以看到，EmployeeOperateAction 继承自 org.apache.struts.actions.DispatchAction，在 EmployeeOperateAction 中定义了多个方法。使用 DispatchAction 的目的在于减少自定义 Action 的数量，从而也可以减少相关的操作。

可以把相关的操作用方法的形式写在同一个 Action 中，这样在 struts-config.xml 文件中处理相关操作的那些请求只用一个<action>元素就可以了。代码如下所示：

```
<action path="/employeeOperateAction"
                scope="request"
                validate="false"
                input="error"
                name="employeeRepastForm"
                type="com.REP.action.EmployeeOperateAction"
```

```
                parameter="method">
                <forward path="repastSuccess" name="RepastSuccess"/>
                <forward path="employeeRegist" name="EmployeeRegist"/>
                <forward path="index" name="Index"/>
                <forward path="banlancesSearch" name="BanlancesSearch"/>
          </action>
```

可以看到 struts-config.xml 文件中定义了一个<action>元素，用来处理员工操作的请求，这样页面中只要发出“employeeOperateAction.do?method=…”这样的请求，就可以由 EmployeeOperateAction 来处理，请求中的 method 值对应了 EmployeeOperateAction 中的具体某个方法。

14.10.3　处理管理员操作请求的 Action

ManagerOperateAction：用于处理餐费管理员相关操作的请求。这个 Action 继承自 DispatchAction，因此在其中包含了一些方法，这些方法分别完成了对管理员操作请求的处理。

```
public class ManagerOperateAction extends DispatchAction {

   IManagerService managerservice;

    //managerLogin 方法用于处理管理员登录验证的请求
   public ActionForward managerLogin(ActionMapping mapping,
            ActionForm form, HttpServletRequest request,
            HttpServletResponse response)   {
        //从页面表单获得管理员输入的用户名和密码
        DynaActionForm dform = (DynaActionForm)form;
        String loginName =dform.getString("loginName");
        String password =dform.getString("password");
        //如果输入的用户名和密码为空，将错误信息返回给输入页面，给出提示
        if(loginName.equals("")||password.equals("")){
            ActionMessages errors = new ActionMessages();
            if(!loginName.equals("")){
                errors.add("password",new ActionMessage("error.password.required"));

            }else if(!password.equals("")){
                errors.add("loginName",new ActionMessage("error.loginName.required"));
            }else{
                errors.add("loginName",new ActionMessage("error.loginName.required"));
                errors.add("password",new ActionMessage("error.password.required"));
            }
            saveErrors(request,errors);
           return    mapping.findForward("ManagerLogin");
        }
        //如果输入格式正确，则检验管理员输入的用户名和密码是否正确
        Boolean    checkresult = managerservice.checkUser(loginName,password);
```

```
            //如果不存在输入的用户名和密码，则将错误信息返回到输入页面，显示出来
            if(!checkresult){
                ActionMessages errors = new ActionMessages();
                //将异常或错误信息存入 ActionMessages 对象 errors 中
                errors.add(ActionMessages.GLOBAL_MESSAGE,new
ActionMessage("error.manangerlogin.managerNotExit"));
                //把 ActionMessages 对象存入到 request 对象中
                saveErrors(request,errors);
                //跳转到错误处理页面
                return mapping.findForward("ManagerLogin");
            }
            //如果输入的用户名和密码正确，则可以跳转到管理员查看和操作员工账户的页面
            return mapping.findForward("GoViewAccount");
        }
      //viewAccount()方法用于取得员工的账户，并将账户列表发送到页面
        public ActionForward viewAccount(ActionMapping mapping,
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response)   {
            List list = managerservice.findAllAccount();
            request.setAttribute("accountList",list);
            return mapping.findForward("ViewAccount");


        }
        //GoModifyAccount()方法用于处理修改账户的请求
        public ActionForward GoModifyAccount(ActionMapping mapping,
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response) throws SQLException {
            String id = request.getParameter("id");
            Account account = managerservice.getAccountById(id);
            String repastCard = account.getLoginName();
            request.setAttribute("id",id);
            request.setAttribute("oldRepastCard",repastCard);
            return mapping.findForward("ModifyAccount");

        }
        //GoFillAccount()方法用于处理跳转到账户充值页面的请求
        public ActionForward GoFillAccount(ActionMapping mapping,
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response) throws SQLException {
            String id = request.getParameter("id");
            request.setAttribute("id",id);
            return mapping.findForward("FillAccount");

        }
        //fillAccount()方法用于处理账户充值的请求
        public ActionForward fillAccount(ActionMapping mapping,
                ActionForm form, HttpServletRequest request,
                HttpServletResponse response) throws SQLException {
```

```
        DynaActionForm dform = (DynaActionForm)form;
        String banlance = dform.getString("banlance");
        String id = request.getParameter("id");
        Account account = managerservice.getAccountById(id);
        try {
            managerservice.fillAccount(account,banlance);
        } catch (AccountNotExistException e) {
            ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,
                    new ActionMessage("error.account.accountNotExist"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
        }
        return mapping.findForward("GoViewAccount");

    }
    // delAccount ()方法用于处理删除账户的请求
    public ActionForward delAccount(ActionMapping mapping,
            ActionForm form, HttpServletRequest request,
            HttpServletResponse response) throws SQLException {

        String id = request.getParameter("id");
        Account account = managerservice.getAccountById(id);
        try {
            managerservice.delAccount(account);
        } catch (AccountNotExistException e) {
            ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,
                    new ActionMessage("error.account.accountNotExist"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
        }

        return mapping.findForward("GoViewAccount");
    }
  // modifyAccount ()方法用于处理修改账户的请求
  public ActionForward modifyAccount(ActionMapping mapping,
            ActionForm form, HttpServletRequest request,
            HttpServletResponse response) throws SQLException {

        DynaActionForm dform = (DynaActionForm)form;
        String repastCard = dform.getString("repastCard");
        String id = request.getParameter("id");
```

```
        try {
            managerservice.modifyAccount(repastCard,id);
        } catch (AccountIsExistException e) {
            ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,new
ActionMessage("error.account.accountIsExist"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
        }

        return mapping.findForward("GoViewAccount");

    }
    public IManagerService getManagerservice() {
        return managerservice;
    }
    public void setManagerservice(IManagerService managerservice) {
        this.managerservice = managerservice;
    }
}
```

14.11　在 Eclipse 中使用 Struts 进行错误处理

在前面介绍的 Action 中处理异常时，多处使用了这样的语句：

```
           ActionMessages errors = new ActionMessages();
            //将异常或错误信息存入 ActionMessages 对象 errors 中
            errors.add(ActionMessages.GLOBAL_MESSAGE,
                      new ActionMessage("error.account.accountNotExist"));
            //把 ActionMessages 对象存入到 request 对象中
            saveErrors(request,errors);
            //跳转到错误处理页面
            return mapping.getInputForward();
```

其中 ActionMessages 对象 errors 可以保存错误信息，Action 中的 saveErrors()方法可以把错误信息保存到 Request 范围内，随着页面的跳转将这些错误信息传递到页面。这些错误信息可以从资源文件中获得，如上面代码中的 error.account.accountNotExist 错误信息，其对应的资源文件中就存在这样的语句：

```
error.account.accountNotExist=账号不存在
```

经过这样的处理，如果错误处理页面使用<html:message>标签，将会接收到错误信息，并将其显示出来。餐费管理系统中定义的错误处理页面 error.jsp 用来接收错误信息并显示出来，页面代码如下所示：

```
<%@ page contentType="text/html;charset=GB2312" language="java" %>
<%@ taglib uri="/WEB-INF/struts-html.tld"   prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-bean.tld"   prefix="bean" %>
<!-- 增加一些注释，使得页面的信息量足够大，
                这样自定义的错误页面才能起作用，否则
                浏览器将显示自身的错误处理页面。这个
                办法在 IE 浏览器起作用，在其他浏览中没
                有做过测试，请读者根据自己所采用的浏览
                器来决定是否使用增加代码信息量的方法。
        -->
<br>
<br>
<h2>您刚访问过的网页出现了如下问题</h2>

 <font color="red">

 <html:messages id="err">
   <li><bean:write name="err" /></li>
 </html:messages>
 </font>
```

代码中加黑的部分用来接收错误信息，并显示错误信息。以刷卡就餐为例，如果输入了错误的就餐卡号，就会转到错误处理页面，显示出相应的错误提示信息，如图 14-17 和图 14-18 所示。



图 14-17　员工刷卡就餐页面

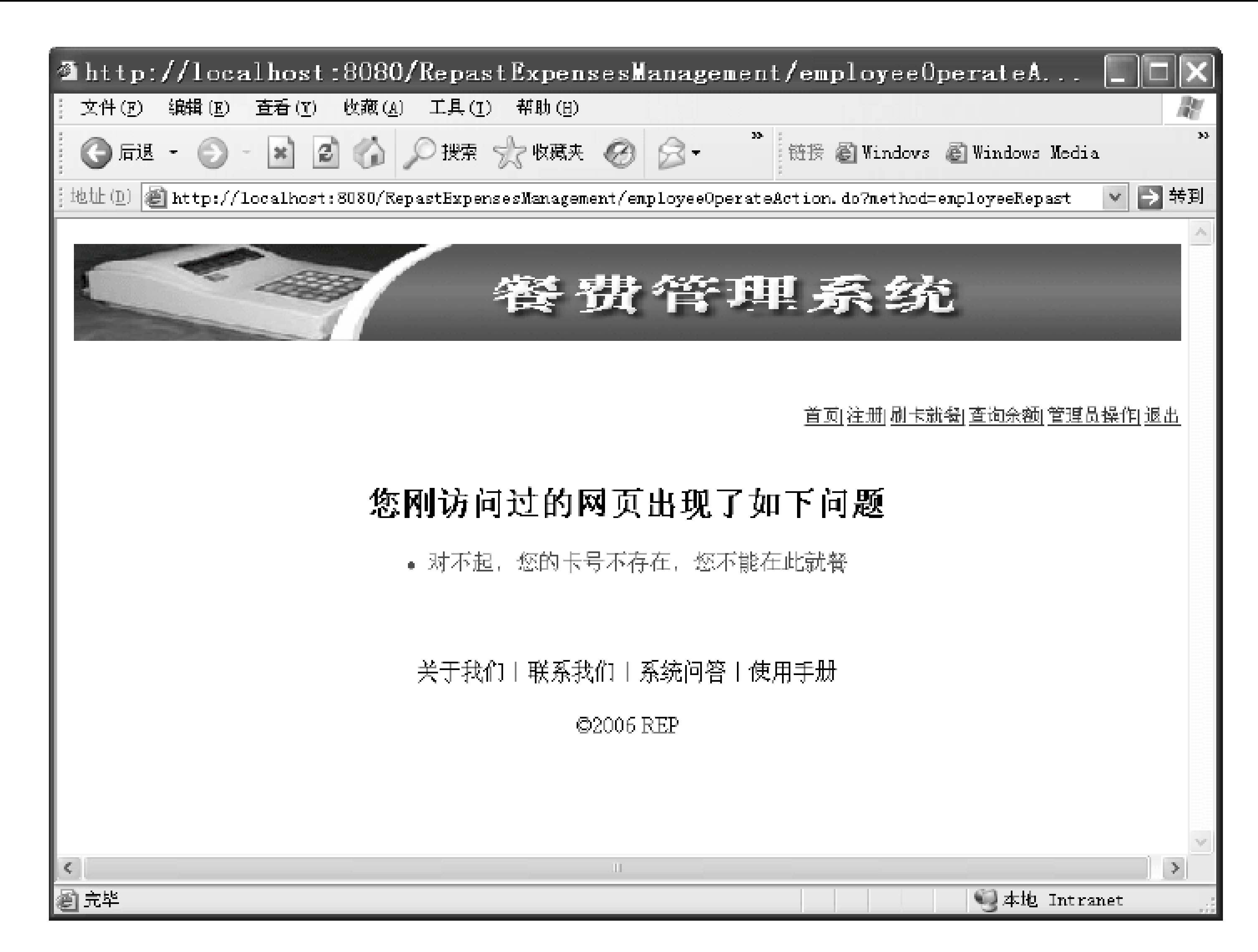


图 14-18　错误处理页面

上面处理的错误信息被保存到 ActionMessages 对象中，代码如下所示：

```
errors.add(ActionMessages.GLOBAL_MESSAGE,
                         new ActionMessage("error.account.accountNotExist"));
```

其中 ActionMessages.GLOBAL_MESSAGE 属于全局常量，因此这样保存的错误信息属于全局错误信息。有时也需要对表单域中的某些部分进行具体错误信息处理，如某一个文本框的输入的错误处理。在 ManagerOperateAction 中的方法 managerLogin()中就针对具体表单输入错误进行了处理，代码如下所示：

```
//如果用户名和密码输入为空，则进行相应的错误处理
if(loginName.equals("")||password.equals("")){
            ActionMessages errors = new ActionMessages();
            if(!loginName.equals("")){
                errors.add("password",new ActionMessage("error.password.required"));

            }else if(!password.equals("")){
                errors.add("loginName",new ActionMessage("error.loginName.required"));
            }else{
                errors.add("loginName",new ActionMessage("error.loginName.required"));
                errors.add("password",new ActionMessage("error.password.required"));
            }
            saveErrors(request,errors);
           return    mapping.findForward("ManagerLogin");
     }
```

从加黑的部分可以看到， ActionMessages 对象中添加的错误信息针对了具体的 loginName 和 password 属性，这两个属性一定要和输入表单中的登录名文本框和密码文本框中的属性相同。经过这样的处理后，如果错误处理页面中使用如下代码：

```
<html:messages id="userNameMsg" property="loginName"/>
<bean:write name="userNameMsg"/>
```

就可以显示出针对登录名 loginName 输入产生的错误信息。图 14-19 和图 14-20 表示不输入登录名和密码从而产生了错误，显示出错误提示信息的过程。

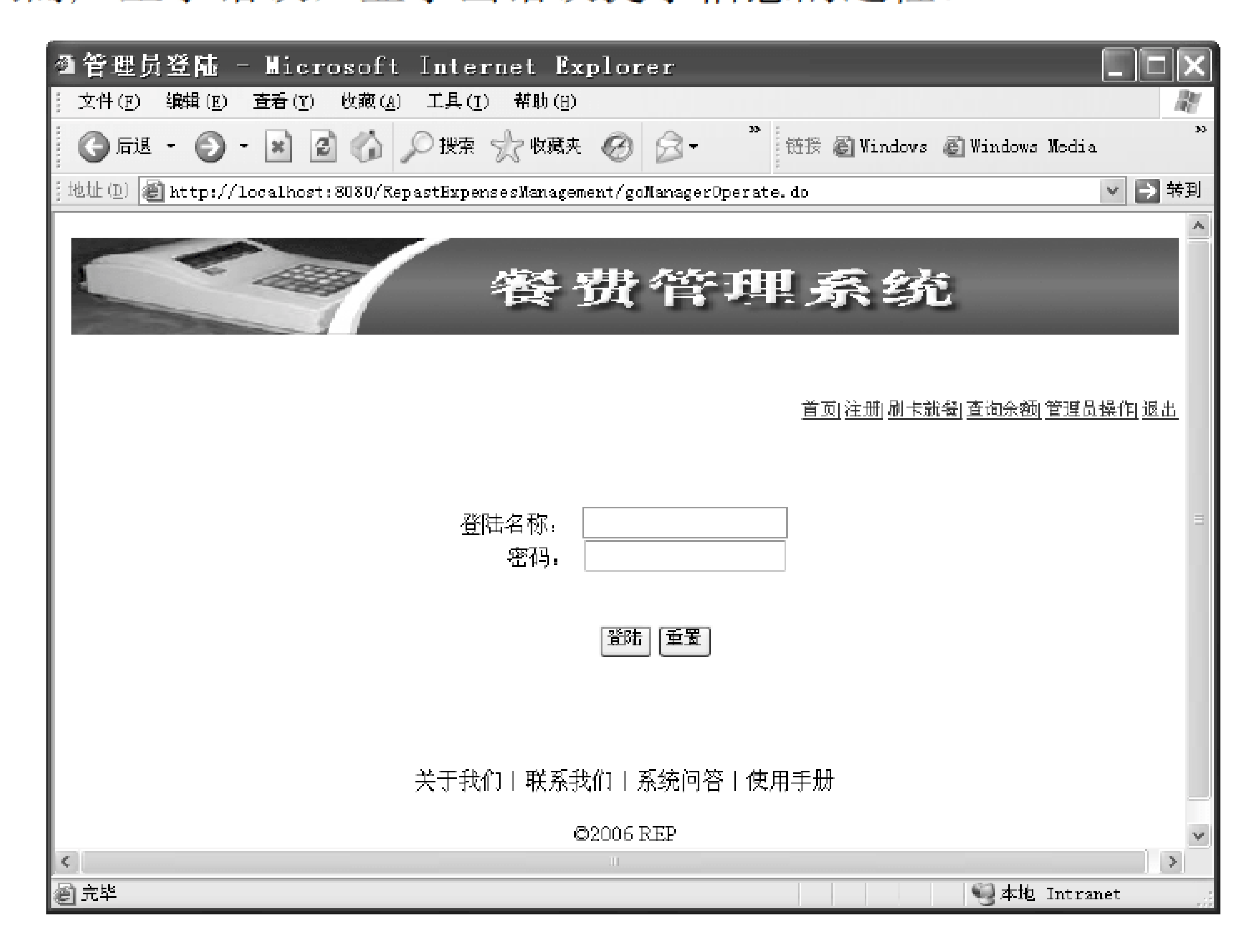


图 14-19　管理员登录页面

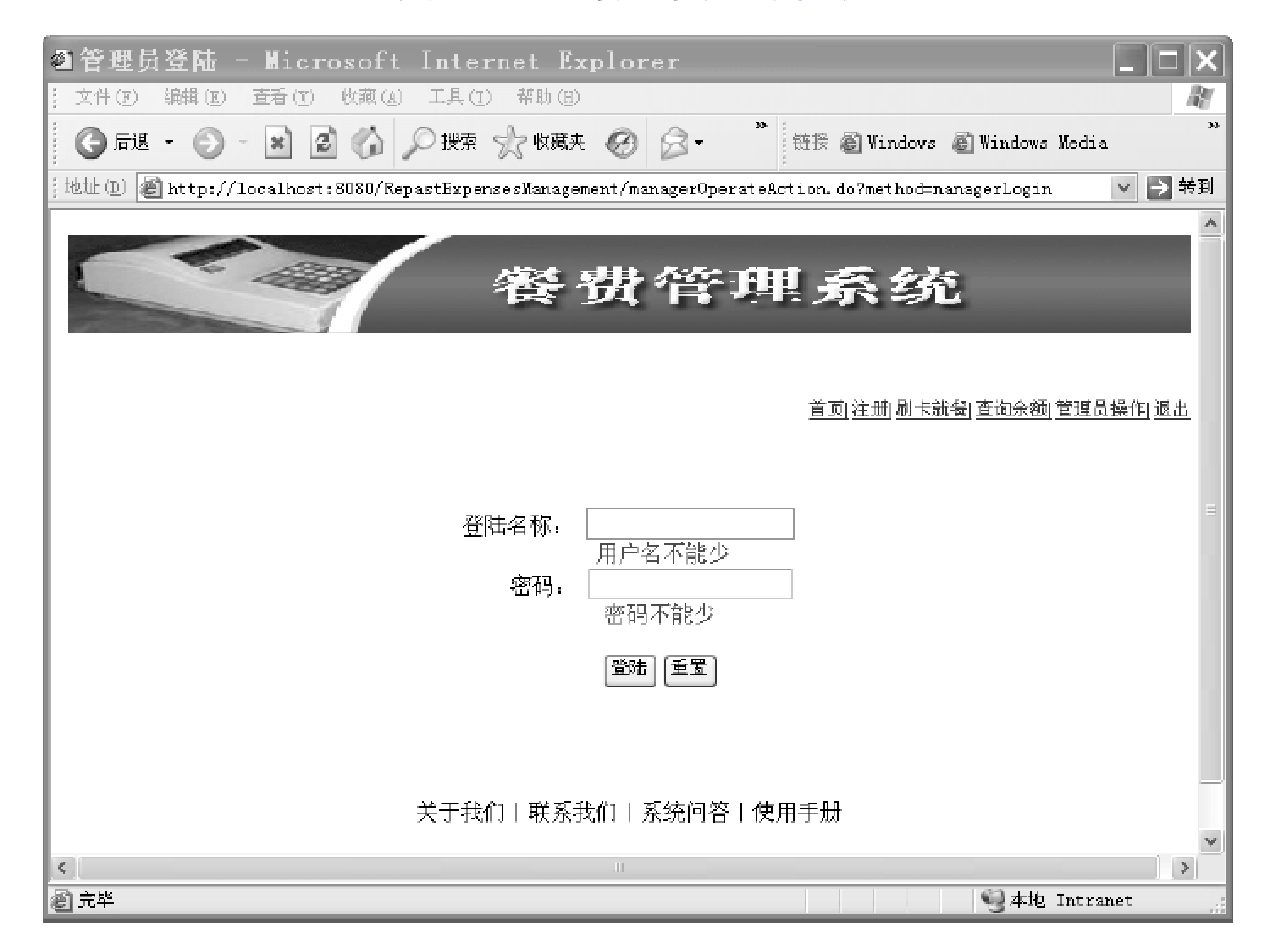


图 14-20　错误显示页面

14.12　在 Eclipse 中建立模型部分

模型部分包括可重复利用的 JavaBean、系统的业务逻辑以及对数据库的访问。本节将重点介绍其中的 JavaBean 的建立，接下来的几节将介绍业务逻辑的建立以及数据库访问的建立。

JavaBean 类存放在项目下的 com.REP.bean 包下，这里的 JavaBean 主要是指业务实体对

象，在餐费管理系统中主要有 3 个业务实体对象，即 Account、Employee、Manager 对象。

这 3 个业务实体类通过 Hibernate Synchronizer 自动生成，关于自动生成的方法读者可查看相关的资料，下面重点介绍 3 个类。

14.12.1　员工账户类

Account 类：表示业务中的员工账户。在后面的 DAO 中通过对 Account 类的操作来实现对数据库中 account 表的操作。代码如下：

```
public class Account extends BaseAccount {
	private static final long serialVersionUID = 1L;

/*构造函数开始*/
	public Account () {
		super();
	}

	/**
	 * 主键作为参数的构造函数
	 */
	public Account (java.lang.Integer id) {
		super(id);
	}

	/**
	 * 各个属性作为参数的构造函数
	 */
	public Account (
		java.lang.Integer id,
		com.REP.bean.Employee employee,
		java.lang.String loginName) {

		super (
			id,
			employee,
			loginName);
	}

/*构造函数结束*/
}
```

可以看到 Account 类继承自 BaseAccount 类，在 Account 类中定义了一些构造函数，而在 BaseAccount 类中则包含了和各个属性对应的 setXXX()方法和 getXXX()方法。BaseAccount 类的内容如下：

```
public abstract class BaseAccount	implements Serializable {

	public static String REF = "Account";
```

```
public static String PROP_LOGIN_NAME = "LoginName";
public static String PROP_EMPLOYEE = "employee";
public static String PROP_OVERDRAW_NUMBER = "OverdrawNumber";
public static String PROP_ID = "Id";
public static String PROP_BALANCE = "Balance";

// 构造方法
public BaseAccount () {
        initialize();
}
/**
 * 主键作为参数的构造方法
 */
public BaseAccount (java.lang.Integer id) {
        this.setId(id);
        initialize();
}
/**
 * 各个属性作为参数的构造方法
 */
public BaseAccount (
        java.lang.Integer id,
        com.REP.bean.Employee employee,
        java.lang.String loginName) {

        this.setId(id);
        this.setEmployee(employee);
        this.setLoginName(loginName);
        initialize();
}
protected void initialize () {}
private int hashCode = Integer.MIN_VALUE;

// 主键
private java.lang.Integer id;

// 属性
private java.lang.String loginName;
private java.math.BigDecimal balance;
private java.lang.Integer overdrawNumber;

// Employee 对象
private com.REP.bean.Employee employee;

//下面是一些 setXXX()和 getXXX()方法
public java.lang.Integer getId () {
        return id;
}
public void setId (java.lang.Integer id) {
```

```
		this.id = id;
		this.hashCode = Integer.MIN_VALUE;
	}
	public java.lang.String getLoginName () {
		return loginName;
	}
	public void setLoginName (java.lang.String loginName) {
		this.loginName = loginName;
	}
	public java.math.BigDecimal getBalance () {
		return balance;
	public void setBalance (java.math.BigDecimal balance) {
		this.balance = balance;
	}

	public java.lang.Integer getOverdrawNumber () {
		return overdrawNumber;
	}

	public void setOverdrawNumber (java.lang.Integer overdrawNumber) {
		this.overdrawNumber = overdrawNumber;
	}

	public com.REP.bean.Employee getEmployee () {
		return employee;
	}

	public void setEmployee (com.REP.bean.Employee employee) {
		this.employee = employee;
	}

	public boolean equals (Object obj) {
		if (null == obj) return false;
		if (!(obj instanceof com.REP.bean.Account)) return false;
		else {
			com.REP.bean.Account account = (com.REP.bean.Account) obj;
			if (null == this.getId() || null == account.getId()) return false;
			else return (this.getId().equals(account.getId()));
		}
	}

	public int hashCode () {
		if (Integer.MIN_VALUE == this.hashCode) {
			if (null == this.getId()) return super.hashCode();
			else {
				String hashStr = this.getClass().getName() + ":" + this.getId().hashCode();
				this.hashCode = hashStr.hashCode();
			}
		}
```

```
		return this.hashCode;
	}


	public String toString () {
		return super.toString();
	}
}
```

BaseAccount 类中定义的属性和数据库中 account 表的字段对应，此外在类中还定义了一个 Employee 对象作为其属性，通过这个属性，在 DAO 中可以通过 Account 对象访问到 Employee 对象，这样 Account 对象和 Employee 对象之间就建立了联系，这个联系称为一对一的关联，这个联系在介绍 DAO 时将会详细介绍。另外在 BaseAccount 类中重写了 equals()、hashCode()、toString()方法。

14.12.2　员工类

Employee 类：用来表示业务中的员工。和 Account 类相同，Employee 类继承自 BaseEmployee 类，在 Employee 类中只是定义了构造函数，关于 Employee 类就不再列举，下面介绍 BaseEmployee 类。

```
public abstract class BaseEmployee  implements Serializable {

	public static String REF = "Employee";
	public static String PROP_ACCOUNT = "account";
	public static String PROP_IDCARD = "Idcard";
	public static String PROP_NAME = "Name";
	public static String PROP_ID = "Id";

	//构造方法
	public BaseEmployee () {
		initialize();
	}
	/**
	 * 主键作为参数的构造方法
	 */
	public BaseEmployee (java.lang.Integer id) {
		this.setId(id);
		initialize();
	}
	/**
	 * 各个属性作为参数的构造方法
	 */
	public BaseEmployee (
		java.lang.Integer id,
		java.lang.String name,
		java.lang.String idcard) {
```

```
        this.setId(id);
        this.setName(name);
        this.setIdcard(idcard);
        initialize();
    }

    protected void initialize () {}
    private int hashCode = Integer.MIN_VALUE;

    // 主键
    private java.lang.Integer id;

    // 主要属性
    private java.lang.String name;

    private java.lang.String idcard;

    // Account 对象作为属性建立一对一关联
    private com.REP.bean.Account account;

    …
    //关于 setXXX()方法和 getXXX()方法省略
}
```

可以看到 Employee 类中定义了一个 Account 对象作为其属性，这样在 DAO 操作中就可以从 Employee 访问到 Account 对象，建立了 Employee 和 Account 对象之间的一对一关联。

14.12.3　管理员类

Manager 类用来表示业务中的管理员。和前面介绍的 Employee 类和 Account 类的定义方法相同，Manager 类继承自 BaseManager 类。下面主要介绍 BaseManager 类：

```
public abstract class BaseManager  implements Serializable {

    public static String REF = "Manager";
    public static String PROP_PASSWORD = "Password";
    public static String PROP_LOGIN_NAME = "LoginName";
    public static String PROP_ID = "Id";

    // 构造方法
    public BaseManager () {
        initialize();
    }
    /**
     * 主键作为参数的构造方法
     */
    public BaseManager (java.lang.Integer id) {
```

```
        this.setId(id);
        initialize();
    }
    /**
     * 各个属性作为参数的构造方法
     */
    public BaseManager (
        java.lang.Integer id,
        java.lang.String loginName,
        java.lang.String password) {

        this.setId(id);
        this.setLoginName(loginName);
        this.setPassword(password);
        initialize();
    }
    protected void initialize () {}
    private int hashCode = Integer.MIN_VALUE;
    // 主键
    private java.lang.Integer id;
    // 主要属性
    private java.lang.String loginName;
    private java.lang.String password;
     …
     //setXXX()方法和 getXXX()方法省略
}
```

Manager 对象并没有和其他对象建立关联，因此在 Manager 类中没有将其他对象作为其属性。在 Manager 类中主要定义用户名 loginName 属性和密码 password 属性。

14.13　在 Eclipse 中建立业务逻辑类

业务逻辑类主要完成应用中所需要的一些业务逻辑，业务逻辑类接口存放在项目下的 com.REP.IService 包下，业务逻辑类主要存放在 com.REP.ServiceImpl 包中。

业务逻辑接口主要有两个，即 IEmployeeService 和 IManagerService，分别对应了和员工相关的业务逻辑以及和餐费管理员相关的业务逻辑。

和业务逻辑接口对应，在系统中定义了实现这两个业务逻辑接口的实现类，即 EmployeeServiceImpl 类和 ManagerServiceImpl 类。下面就这些内容分别进行介绍。

14.13.1　员工业务逻辑

1．员工业务逻辑接口

IEmployeeService：员工业务逻辑接口。在这个接口中定义了一系列的方法，用于实现

员工的业务操作。代码如下：

```
 public interface IEmployeeService {
    //以 Account 对象为参数的员工注册业务逻辑
    public void regist(Account account) throws AccountIsExistException;

    //以就餐账户名称、员工姓名、员工身份证号为参数的注册业务逻辑
    public void registByName(String repastCard,String employeeName,String idCard)
    throws AccountIsExistException, EmployeeNotExistException,EmployeeBeUsedException;

    //无返回值的员工就餐刷卡业务逻辑
    public void repast(String name,String fee)throws AccountNotExistException, OverDraw
Exception;

    //返回 Account 对象的就餐刷卡业务逻辑
    public Account   repastAccount(String name,String fee)
    throws AccountNotExistException, OverDrawException ;

    //以就餐账户名称为参数的余额查询业务逻辑
    public BigDecimal searchBanlances(String repastCard) throws AccountNotExistException;

    //以 Account 对象为参数的余额查询业务逻辑
    public BigDecimal searchBanlancesByAccount(Account account);

    //以 Account 对象为参数的透支次数查询业务逻辑
    public Integer searchOverDrawNubByAccount(Account account);

    //以就餐账户名称为参数的透支次数查询业务逻辑
    public Integer searchOverDrawNub(String repastCard) throws AccountNotExistException;
}
```

2．员工业务逻辑实现

EmployeeServiceImpl：员工业务逻辑实现。这个类实现了 IEmployeeService 接口。在 Action 中可以使用这个类来处理来自员工的操作请求。代码如下：

```
public class EmployeeServiceImpl implements IEmployeeService {
   //定义 DAO 对象
    private IEmployeeDAO employeedao ;
    private IAccountDAO accountdao ;
    //以 Account 对象为参数的员工注册业务逻辑
    public void regist(Account account) throws AccountIsExistException {
        //根据账户名称来获得 Account 对象
        Account acnt = accountdao.getAccountbyName(account.getLoginName());
        //判断是否已经存在同名账户，如果存在抛出异常
        if(!acnt.equals(null)) throw new AccountIsExistException();
        //如果账户不存在，对此账户进行保存，完成注册业务
        accountdao.save(account);

    }
```

```
  //以就餐账户名称、员工姓名、员工身份证号为参数的注册业务逻辑
    public void registByName(String repastCard,String employeeName,String idCard) throws
AccountIsExistException, EmployeeNotExistException, EmployeeBeUsedException {
        //根据员工姓名和身份证号查找数据库中是否存在该名员工
        Employee employee = employeedao.findEmployee(employeeName,idCard);
        //如果员工不存在，则抛出异常，不允许注册
        if(employee==null) throw new EmployeeNotExistException();
        //如果员工存在，接下来要判断注册的账户名是否已存在
        Account acnt = accountdao.getAccountbyName(repastCard);
        //如果存在，抛出异常，不允许注册
        if(!(acnt==null)) throw new AccountIsExistException();
        //如果账户不存在，查找该员工是否已经注册过
        Account account = employee.getAccount();
        //如果注册过，则不允许多次注册，抛出异常
        if(!(account==null))throw new EmployeeBeUsedException();
        //如果员工没有注册过，允许注册
        account = new Account();
        account.setBalance(new BigDecimal("0"));
        account.setLoginName(repastCard);
        account.setOverdrawNumber(0);
        account.setEmployee(employee);
        accountdao.save(account);


    }
    //无返回值的员工就餐刷卡业务逻辑
    public void repast(String name,String fee) throws AccountNotExistException, OverDraw
Exception {
        Account account = accountdao.getAccountbyName(name);
        if(account==null) throw new AccountNotExistException();
        Integer overdrawNub = account.getOverdrawNumber();
        if(overdrawNub.equals(3))throw new OverDrawException();
        BigDecimal balances = account.getBalance();
        BigDecimal repastFee = new BigDecimal(fee);
        if(balances.compareTo(repastFee)<=0){
            balances = balances.subtract(repastFee);
            account.setBalance(balances);

            overdrawNub = overdrawNub+1;
            account.setOverdrawNumber(overdrawNub);

            accountdao.update(account);
        }else{
            balances = balances.subtract(repastFee);
            account.setBalance(balances);
            accountdao.update(account);
        }
    }
    //返回 Account 对象的就餐刷卡业务逻辑
```

```
public Account   repastAccount(String name,String fee)
                        throws AccountNotExistException, OverDrawException {
    //根据账户名称获得就餐账户对象
    Account account = accountdao.getAccountbyName(name);
    //如果账户不存在，不允许就餐，抛出异常
    if(account==null) throw new AccountNotExistException();
    //如果账户存在，查看透支次数
    Integer overdrawNub = account.getOverdrawNumber();
    //如果透支次数超过 3 次，不允许就餐，抛出异常
    if(overdrawNub.equals(3))throw new OverDrawException();
    //如果透支次数没有超过 3 次，允许就餐，获得余额和就餐标准
    BigDecimal balances = account.getBalance();
    BigDecimal repastFee = new BigDecimal(fee);
    //如果就餐标准超过余额，则余额减少的同时需要增加一次透支次数
    if(balances.compareTo(repastFee)<=0){
        balances = balances.subtract(repastFee);
        account.setBalance(balances);

        overdrawNub = overdrawNub+1;
        account.setOverdrawNumber(overdrawNub);

        accountdao.update(account);
    }else{
        //否则只需减少余额
        balances = balances.subtract(repastFee);
        account.setBalance(balances);
        accountdao.update(account);
    }
    return account;
}
//以就餐账户名称为参数的余额查询业务逻辑
public BigDecimal searchBanlances(String repastCard) throws AccountNotExistException{
    Account account = accountdao.getAccountbyName(repastCard);
    if(account==null) throw new AccountNotExistException();
    BigDecimal banlances = account.getBalance();
    return banlances;
}
//以 Account 对象为参数的余额查询业务逻辑
public BigDecimal searchBanlancesByAccount(Account account){
    BigDecimal banlances = account.getBalance();
    return banlances;
}
//以就餐账户名称为参数的透支次数查询业务逻辑
public Integer searchOverDrawNub(String repastCard) throws AccountNotExistException{
    Account account = accountdao.getAccountbyName(repastCard);
    if(account==null) throw new AccountNotExistException();
    Integer overDrawNub = account.getOverdrawNumber();
    return overDrawNub;
}
```

```
    //以 Account 对象为参数的透支次数查询业务逻辑
    public Integer searchOverDrawNubByAccount(Account account){
        Integer overDrawNub = account.getOverdrawNumber();
        return overDrawNub;
    }
    public IAccountDAO getAccountdao() {
        return accountdao;
    }
    public void setAccountdao(IAccountDAO accountdao) {
        this.accountdao = accountdao;
    }
    public IEmployeeDAO getEmployeedao() {
        return employeedao;
    }
    public void setEmployeedao(IEmployeeDAO employeedao) {
        this.employeedao = employeedao;
    }
}
```

从代码中可以看到，业务逻辑中需要一些对数据库的访问，这些访问用 DAO 对象来完成，在 EmployeeServiceImpl 类中定义了 IEmployeeDAO 对象和 IAccountDAO 对象，这两个对象分别用来完成对于员工数据的访问和账户数据的访问。

在 EmployeeServiceImpl 类中通过 setXXX()方法和 getXXX()方法来完成 DAO 对象的初始化，这样就可以在 Spring 的配置文件中用配置的方式定义这些 DAO 对象和 EmployeeServiceImpl 对象的依赖关系，并实现在运行期将这种依赖注入。

14.13.2　管理员业务逻辑

1．管理员业务逻辑接口

IManagerService：管理员业务逻辑接口。在这个接口中定义了与管理员相关的业务操作。代码如下：

```
public interface IManagerService {
    //删除账户
    public void delAccount(Account account)throws AccountNotExistException;
    //以 Account 对象为参数修改账户
    public void modifyAccount(Account account)throws AccountNotExistException;
    //以就餐账户名称和账户 id 为参数修改账户
    public void modifyAccount(String repastCard ,String id) throws   AccountIsExistException;
    //账户充值
    public void fillAccount(Account account,String fee)throws AccountNotExistException;
    //检查用户是否存在
    public Boolean checkUser(String username,String password);
    //获得所有账户
    public List findAllAccount();
    //通过账户 id 获得账户
    public Account getAccountById(String id);
```

```
        public IAccountDAO getAccountdao() ;
        public void setAccountdao(IAccountDAO accountdao) ;

}
```

2．管理员业务逻辑实现

ManagerServiceImpl：管理员业务逻辑实现。在这个类中实现了 IManagerService 接口中定义的方法，在 Action 中可以通过这个类来处理来自管理员的操作请求。代码如下：

```
public class ManagerServiceImpl implements IManagerService {
        //定义 DAO 对象
        public IAccountDAO accountdao   ;
        public IManagerDAO managerdao;

        public IAccountDAO getAccountdao() {
                return accountdao;
        }

        public void setAccountdao(IAccountDAO accountdao) {
                this.accountdao = accountdao;
        }
        //删除账户
        public void delAccount(Account account) throws AccountNotExistException {
                    if(account == null)throw new AccountNotExistException();
                     accountdao.delete(account);
        }
        //以 Account 对象为参数修改账户
        public void modifyAccount(Account account) throws AccountNotExistException {
                if(account==null)throw new AccountNotExistException();
                accountdao.update(account);

        }
        //以就餐账户名称和账户 id 为参数修改账户
        public void modifyAccount(String repastCard ,String id) throws   AccountIsExistException {
                Account account = new Account();
                //根据账户名称获得账户对象
                account = accountdao.getAccountbyName(repastCard);
                //如果账户不存在，则抛出异常
                if(!(account==null)) throw new AccountIsExistException();
                //如果账户存在，对账户的名称进行修改
                account = getAccountById(id);
                account.setLoginName(repastCard);
                accountdao.update(account);

        }
        //账户充值
        public void fillAccount(Account account,String fee) throws AccountNotExistException {
                //如果账户为空，则无法充值，抛出异常
                if(account==null)throw new AccountNotExistException();
```

```
            //如果账户存在，获得账户余额
            BigDecimal banlances = account.getBalance();
            Integer overDrawNub = account.getOverdrawNumber();
            //增加账户余额
            banlances =banlances.add(new BigDecimal(fee));
            //如果账户余额大于 0，则将透支次数设为 0
            if(banlances.compareTo(new BigDecimal("0"))>=0){
                overDrawNub = 0;
            }
            //更新账户
            account.setBalance(banlances);
            account.setOverdrawNumber(overDrawNub);
            accountdao.update(account);

        }
        //检查用户是否存在
        public Boolean checkUser(String username, String password) {
            Manager manager = managerdao.findUser(username,password);
            if(manager==null) return false;
            return true;
        }

        public void setManagerdao(IManagerDAO managerdao) {
            this.managerdao = managerdao;
        }
        //获得所有账户
        public List findAllAccount(){
            List list = new ArrayList();
            list = accountdao.findAll();
            return list;
        }
        //根据账户 id 获得账户
        public Account getAccountById(String id){
            Account account = accountdao.get(Integer.valueOf(id));
            return account;
        }
}
```

14.14　在 Eclipse 中使用 Hibernate 建立 DAO 类

在前面的业务逻辑实现中多处用到了 DAO 对象，DAO 对象主要完成对数据库的访问。传统对数据库的访问通过 JDBC 来完成，但是使用 JDBC 来访问数据库一方面 SQL 语句比较难以理解，比较复杂；同时访问程序中重复的代码较多。

Hibernate 可以解决这些问题。Hibernate 通过访问对象的方法来实现对数据库的访问，在这种访问中使用 HQL 语句来访问对象，使得程序变得非常简单，同时 Hibernate 将数据源的各种参数定义在配置文件中，因此开发者不需要再写获得数据源、建立数据库连接这

样的语句，使开发变得简单。

当然如果想通过访问业务实体对象来实现访问数据库，必须建立业务实体对象和数据库联系的桥梁，这个桥梁就是对象-关系映射文件。下面首先介绍与各个业务实体对象对应的对象-关系映射文件，然后介绍应用中 DAO 类的建立。

14.14.1　建立对象-关系映射文件

在餐费管理系统中根据业务实体对象和数据库中的表，需要建立 3 个对象-关系映射文件。

- Account.hbm.xml：表示 Account 对象与 account 表之间的对象-关系映射文件。
- Employee.hbm.xml：表示 Employee 对象和 employee 表之间的对象-关系映射文件。
- Manager.hbm.xml：表示 Manager 对象和 manager 表之间的对象-关系映射文件。

下面分别对这 3 个对象-关系映射文件进行介绍。

1．Account.hbm.xml 对象-关系映射文件

在这个映射文件中定义了 Account 类和 account 表的对应关系。具体表现在 Acount 类的属性和 account 表的字段的对应。代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping 3.0 DTD//EN"
	"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping package="com.REP.bean">
	<class
		name="Account"
		table="account"
	>
		<meta attribute="sync-DAO">true</meta>
		<id
			name="Id"
			type="integer"
			column="id"
		>
			<generator class="increment"/>
		</id>
		<property
			name="LoginName"
			column="loginName"
			type="string"
			not-null="true"
			length="20"
		/>
		<property
			name="Balance"
			column="balance"
```

```
                type="big_decimal"
                not-null="false"
                length="5"
            />
            <property
                name="OverdrawNumber"
                column="overdrawNumber"
                type="integer"
                not-null="false"
                length="1"
            />
        <!--定义与 Employee 对象的一对一关系-->
         <many-to-one name="employee"
          class="com.REP.bean.Employee"
          column="employee_id"
          not-null="true"
          unique="true"
          cascade="save-update"
          />
    </class>
</hibernate-mapping>
```

在代码中，<property>元素表示了业务实体对象的属性和数据库中字段的对应，其中 name 属性值表示业务实体对象的属性，column 属性值表示数据库中的字段。<many-to-one>元素表示 Account 对象和 Employee 对象建立了关联，其中 unique 属性值为 true，表示 Account 对象和 Employee 对象之间是一对一的关系。

2．Employee.hbm.xml 对象-关系映射文件

在这个映射文件中定义了 Employee 类和 employee 表的对应关系。具体表现在 Employee 类的属性和 employee 表的字段的对应。代码如下：

```
    <?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping 3.0 DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

<hibernate-mapping package="com.REP.bean">
    <class
        name="Employee"
        table="employee"
    >
        <meta attribute="sync-DAO">true</meta>
        <id
            name="Id"
            type="integer"
            column="id"
        >
            <generator class="increment"/>
        </id>
```

```
          <property
                name="Name"
                column="name"
                type="string"
                not-null="true"
                length="20"
          />
          <property
                name="Idcard"
                column="idcard"
                type="string"
                not-null="true"
                length="18"
          />

        <one-to-one name="account"
          class="com.REP.bean.Account"
          cascade="all"
          property-ref="employee"
          />

        </class>
</hibernate-mapping>
```

在这个配置文件中，用<one-to-one>元素建立了 Employee 对象和 Account 对象之间的关联，同时也表示两个对象之间是一对一的关系。

3．Manager.hbm.xml 对象-关系映射文件

在这个映射文件中定义了 Manager 类和 manager 表的对应关系。具体表现在 Manager 类的属性和 manager 表的字段的对应。代码如下：

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
        "-//Hibernate/Hibernate Mapping 3.0 DTD//EN"
        "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.REP.bean">
        <class
                name="Manager"
                table="manager"
        >
                <meta attribute="sync-DAO">true</meta>
                <id
                        name="Id"
                        type="integer"
                        column="id"
                >
                        <generator class="increment"/>
                </id>
```

```
          <property
                name="LoginName"
                column="loginName"
                type="string"
                not-null="true"
                length="20"
          />
          <property
                name="Password"
                column="password"
                type="string"
                not-null="true"
                length="20"
          />
     </class>
</hibernate-mapping>
```

经过对象-关系映射文件的建立，在进行数据访问时就可以通过访问业务实体对象来访问数据库了。

14.14.2　建立 DAO 类

餐费管理系统中的数据访问类存在于 com.REP.DAO 中，通过 Hibernate Synchronizer 自动生成，包括了 EmployeeDAO 类、AccountDAO 类、ManagerDAO 类。这 3 个类都分别实现了接口 IEmployeeDAO、IAccountDAO、IManagerDAO。

同时每个 DAO 类还分别继承自 BaseEmployeeDAO、BaseAccountDAO、BaseManagerDAO 类，这几个 Base DAO 类共同继承自_BaseRootDAO，关于 DAO 的继承关系可以参考本书前面章节。

_BaseRootDAO 类是一个抽象类，在其中定义了大量的数据访问方法，包括 find()、load()、save()、delete()等，用户可以通过继承类直接使用这些方法，或者重写这些方法。关于这些方法读者可以查看源程序，_BaseRootDAO 位于 com.REP.DAO.base 路径下。接下来主要介绍在餐费管理系统中的 DAO 接口和相应的 DAO 实现。

1．员工数据访问接口

在这个接口中主要定义需要对 employee 表进行数据访问的一些方法，包括保存、删除、查找、更新等。代码如下：

```
public interface IEmployeeDAO {
     public com.REP.bean.Employee get(java.lang.Integer key);
     public com.REP.bean.Employee load(java.lang.Integer key);
     public java.util.List<com.REP.bean.Employee> findAll ();
     public java.lang.Integer save(com.REP.bean.Employee employee);
     public void saveOrUpdate(com.REP.bean.Employee employee);
     public void update(com.REP.bean.Employee employee);
     public void delete(java.lang.Integer id);
```

```
        public void delete(com.REP.bean.Employee employee);
        public Employee findEmployee(String employeeName,String idCard);
}
```

在这些方法中，findEmployee()方法是自定义的一个方法，用于查找员工。其他方法由 Hibernate Synchronizer 自动生成。

2. 员工数据访问实现

EmployeeDAO：IEmployeeDAO 的实现类，对数据库中的 employee 表进行的数据访问，这里主要说明了 findEmployee()方法的实现。这个方法通过参数 employeeName（员工姓名）以及 idCard（身份证号）来查找表 employee 中是否存在符合条件的员工。代码如下：

```
public class EmployeeDAO extends BaseEmployeeDAO implements com.REP.DAO.iface.
IEmployeeDAO {

        public EmployeeDAO () {}

        public EmployeeDAO (Session session) {
                super(session);
        }
       //查找员工
       public Employee findEmployee(String employeeName,String idCard){
              //定义查找字符串
              String hql = "from Employee as e where e.Name=:employeename and e.Idcard=:idcard";
                //执行查找
              Query query = getQuery(hql,getSession());
              query.setString("employeename",employeeName);
              query.setString("idcard",idCard);
              List list = new ArrayList();
              list = query.list();
              if(list.isEmpty()) return null;
              return (Employee)list.iterator().next();
       }
}
```

3. 账户数据访问接口

IAccountDAO：在这个接口中主要定义需要对 employee 表进行数据访问的一些方法，包括保存、删除、查找、更新等。代码如下：

```
public interface IAccountDAO {
        public com.REP.bean.Account get(java.lang.Integer key);
        public com.REP.bean.Account load(java.lang.Integer key);
        public java.util.List<com.REP.bean.Account> findAll ();
        public java.lang.Integer save(com.REP.bean.Account account);
        public void saveOrUpdate(com.REP.bean.Account account);
        public void update(com.REP.bean.Account account);
        public void delete(java.lang.Integer id);
        public void delete(com.REP.bean.Account account);
         public Account getAccountbyName(String name);
}
```

4. 账户数据访问实现

AccountDAO：IAccountDAO 的实现类，对数据库中的 account 表进行的数据访问。这里主要说明了 getAccountByName()方法的实现过程。这个方法用来通过员工的账户名称获得员工账户 Account 对象。代码如下：

```
public class AccountDAO extends BaseAccountDAO implements com.REP.DAO.iface.IAccount
DAO {

      public AccountDAO () {}

      public AccountDAO (Session session) {
            super(session);
      }
     //重写了 BaseAccountDAO 的 findFiletered()方法，用于查找符合条件的记录
     protected Criteria findFiltered (Session s, String propName, Object filter) {
           Criteria crit = s.createCriteria(getReferenceClass());
           crit.add(Expression.eq(propName, filter));
           return crit;
     }
     //根据账户名称获得账户
     public Account getAccountbyName(String name) {
           List list = new ArrayList();
           //获得账户列表
           list = findFiltered(getSession(),"LoginName",name).list();
           //如果列表非空，获得账户，并返回
           if(!list.isEmpty()){
                 Account account =(Account)list.iterator().next();
                 closeCurrentSession();
                 return account;
           }else{
                 closeCurrentSession();
                 return null;
           }
     }
}
```

5. 管理员数据访问接口

IManagerDAO：管理员数据访问接口，定义了对管理员表 manager 进行的一些数据访问操作。代码如下：

```
public interface IManagerDAO {
      public com.REP.bean.Manager get(java.lang.Integer key);
      public com.REP.bean.Manager load(java.lang.Integer key);
      public java.util.List<com.REP.bean.Manager> findAll ();
      public java.lang.Integer save(com.REP.bean.Manager manager);
      public void saveOrUpdate(com.REP.bean.Manager manager);
      public void update(com.REP.bean.Manager manager);
      public void delete(java.lang.Integer id);
      public void delete(com.REP.bean.Manager manager);
```

```
	public Manager findUser(String username,String password);
}
```

6．管理员数据访问实现

ManagerDAO：IManagerDAO 的实现类，对数据库中的 manager 表进行的数据访问。在这里主要说明了 findUser()方法的实现过程。这个方法根据参数：用户名（username）和密码（password）来查找 manager 表中是否存在符合条件的记录。如果得到返回值将获返回 Manager（管理员）对象：

```
public class ManagerDAO extends BaseManagerDAO implements com.REP.DAO.iface.IManager
DAO {

	public ManagerDAO () {}

	public ManagerDAO (Session session) {
		super(session);
	}
	//查找用户是否存在,如果存在返回 Manager 对象，如果不存在返回 null
	public Manager findUser(String username,String password){
		List list = new ArrayList();

		Manager manager = new Manager();

		String hql = "from Manager"
			+ " as u where u.LoginName=:username and u.Password=:password";

		Query query = getQuery(hql, getSession());
		query.setString("username",username);
		query.setString("password",password);

		list = query.list();
		if(!list.isEmpty()){
			manager =(Manager)list.iterator().next();
			closeCurrentSession();
			return manager;
		}
		return null;
	}

}
```

14.15　在 Eclipse 中使用 Spring 装配各个组件

前面已经介绍了用于控制业务流程的 Action 类，用于实现业务逻辑的 Service 类，以及用户数据访问的 DAO 类。这些类在传统的做法中，通常在代码中通过互相调用组织在一起，在编译期就耦合在了一起，一旦某些代码发生了改变，就需要对耦合在一起的程序进行重

新编译，这样维护起来就比较困难。

Spring 框架的依赖注入解决了这个问题，Spring 把应用中的各个组件通过配置文件组织在了一起，各个组件之间的依赖关系在运行期注入，这样，如果某些代码发生了改变就不需要对所有的代码全部重新编译，提高了维护的效率，同时也降低了组件之间的耦合程度。

Struts 能够和 Spring 进行很好的结合，结合后 Struts 中的 Action 就可以被 Spring 进行管理，从而也实现了在运行期对 Aciton 的依赖注入。

14.15.1　Struts 和 Spring 的集成

Struts 和 Spring 结合需要完成以下的工作。

（1）在 Struts 的配置文件中添加 Spring 插件，代码如下。

```
    <!-- 配置 Spring 插件-->
<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
    <set-property property="contextConfigLocation" value="/WEB-INF/applicationContext.xml" >
              </set-property>
 </plug-in>
```

（2）修改 RequestProcessor，代码如下。

```
public class EncodingProcessor extends DelegatingTilesRequestProcessor {
    public void process(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException {

        request.setCharacterEncoding("UTF-8");
        response.setContentType("text/html;charset=UTF-8");
        super.process(request, response);
    }

}
```

RequestProcessor 被用来在 Struts 进行请求的处理，EncodingProcessor 属于自定义的 RequestProcessor，这个类继承了 Spring 框架的 DelegatingTilesRequestProcessor，这样当处理用户请求需要用到 Action 时，就需要转移控制权，到 Spring 的配置文件中去寻找相应的 Action 来进行处理。

（3）在 Spring 中配置处理各种请求的 Action。

这部分内容在 Spring 的配置文件 applicationContext 文件中进行配置，接下来将会详细介绍。

除了能够实现运行期的依赖注入，Spring 框架的面向方面的编程方式（AOP）也非常有用，在 AOP 的基础上很容易实现事务的管理。

14.15.2　建立 applicationContext.xml

在餐费管理系统中，Spring 中的配置文件是 applicationContext.xml，这个文件存在于

\WEB-INF 路径下。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
                                        "http://www.springframework.org/dtd/spring-beans.dtd">

<beans>
    <!--定义数据源-->
    <bean id="dataSource" class="org.springframework.jdbc.datasource.DriverManagerData
Source">
        <property name="driverClassName">
            <value>com.mysql.jdbc.Driver</value>
        </property>
        <property name="url">

    <value>jdbc:mysql://localhost:3306/rep?useUnicode=true&amp;characterEncoding=UTF-8</
value>
        </property>
        <property name="username">
            <value>root</value>
        </property>
        <property name="password">
            <value>root</value>
        </property>
    </bean>
   <!--定义 sessionFactory-->
    <bean id="sessionFactory" class="org.springframework.orm.hibernate3.LocalSessionFactory
Bean" >
       <property name="dataSource">
           <ref bean="dataSource"/>
       </property>
       <property name="mappingResources">
           <list>
                <value>com/REP/hbm/Account.hbm.xml</value>
                 <value>com/REP/hbm/Employee.hbm.xml</value>
                 <value>com/REP/hbm/Manager.hbm.xml</value>
            </list>
       </property>
       <property name="hibernateProperties">
           <props>
               <prop key="hibernate.dialect">
                   org.hibernate.dialect.MySQLDialect
               </prop>
                  <prop key="hibernate.show_sql">true</prop>
           </props>
       </property>
  </bean>

    <!--定义账户-->
    <bean id="Account" class="com.REP.bean.Account"/>
```

```
<!--定义员工-->
<bean id="Employee" class="com.REP.bean.Employee"/>
<!--定义管理员-->
<bean id="Manager" class="com.REP.bean.Manager"/>

<!--定义各个 DAO-->
<bean id="accountDAO" class="com.REP.DAO.AccountDAO">
    <property name="sessionFactory">
        <ref local="sessionFactory"></ref>
    </property>
</bean>
<bean id="employeeDAO" class="com.REP.DAO.EmployeeDAO">
    <property name="sessionFactory">
        <ref local="sessionFactory"></ref>
    </property>
</bean>
<bean id="managerDAO" class="com.REP.DAO.ManagerDAO">
    <property name="sessionFactory">
        <ref local="sessionFactory"></ref>
    </property>
</bean>


<!--定义业务逻辑-->
<bean id="employeeService" class="com.REP.ServiceImpl.EmployeeServiceImpl">
    <property name="employeedao">
            <ref local="employeeDAO"></ref>
      </property>
      <property name="accountdao">
            <ref local="accountDAO"></ref>
      </property>
</bean>

<bean id="managerService" class="com.REP.ServiceImpl.ManagerServiceImpl">
    <property name="managerdao">
            <ref local="managerDAO"></ref>
      </property>
      <property name="accountdao">
            <ref local="accountDAO"></ref>
      </property>
</bean>

<!--定义事务管理器-->
<bean id="transactionManager"
     class="org.springframework.orm.hibernate3.HibernateTransactionManager">
      <property name="sessionFactory">
            <ref bean="sessionFactory"/>
       </property>
</bean>
```

```
<!--定义产品业务逻辑事务代理-->
<bean id="employeeServiceProxy"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
        <property name="proxyInterfaces">
                <value>com.REP.IService.IEmployeeService</value>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager" />
        </property>
        <property name="target">
            <ref local="employeeService" />
        </property>
        <property name="transactionAttributes">
                <props>
                    <prop key="re*">PROPAGATION_REQUIRED</prop>
                    <prop key="search*">PROPAGATION_REQUIRED,readOnly
</prop>
                </props>
        </property>
</bean>

<bean id="managerServiceProxy"
    class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
        <property name="proxyInterfaces">
                <value>com.REP.IService.IManagerService</value>
        </property>
        <property name="transactionManager">
            <ref bean="transactionManager" />
        </property>
        <property name="target">
            <ref local="managerService" />
        </property>
        <property name="transactionAttributes">
                <props>
                    <prop key="del*">PROPAGATION_REQUIRED</prop>
                    <prop key="modi*">PROPAGATION_REQUIRED</prop>
                    <prop key="fill*">PROPAGATION_REQUIRED</prop>
                    <prop key="check*">PROPAGATION_REQUIRED,readOnly
</prop>
                </props>
        </property>
</bean>

<!--配置处理员工操作请求的 Action-->
<bean name="/employeeOperateAction" class="com.REP.action.EmployeeOperateAction"
     singleton="false">
    <property name="employeeservice">
        <ref local="employeeServiceProxy"></ref>
```

```
            </property>
        </bean>
        <!--配置处理员工注册的 Action-->
       <bean name="/employeeRegist" class="com.REP.action.EmployeeRegistAction"
             singleton= "false">
            <property name="employeeservice">
                <ref local="employeeServiceProxy"></ref>
            </property>
        </bean>
        <!--配置处理管理员操作的 Action-->
        <bean name="/managerOperateAction" class="com.REP.action.ManagerOperateAction"
             singleton="false">
            <property name="managerservice">
                <ref local="managerServiceProxy"></ref>
            </property>
        </bean>
</beans>
```

可以看到在配置文件中通过<bean>元素将各个组件装配起来。在配置文件中定义了以下内容：

- 数据源：和数据库相关的参数在数据源中被定义。
- SessionFactory：用来初始化 Hibernate，创建 Session 对象，通过 Session 对象来保存、更新、删除、加载和查询对象，DAO 的操作需要 Session 来完成，因此 DAO 依赖 SessionFactory，在定义 DAO 时需要引入 SessionFactory 对象的依赖。在 SessionFactory 的配置中，对象-关系映射文件也在其中。
- 事务管理器：通过事务管理器，可以将事务的具体操作委托给其他事务管理机制，如 Hibernate 的事务管理机制。
- 业务逻辑事务代理：org.springframework.transaction.interceptor.TransactionProxyFactoryBean 类用来完成事务代理，配置文件对各种业务逻辑添加了事务代理，这样当访问到业务逻辑中的某些方法，如下面代码中<prop>元素定义的 key 属性值表示的方法时，就会执行<prop>元素中的事务，如 PROPAGATION_REQUIRED。

```
<props>
            <prop key="re*">PROPAGATION_REQUIRED</prop>
            <prop key="search*">PROPAGATION_REQUIRED,readOnly</prop>
</props>
```

- 各个业务实体对象。
- 各个 DAO 对象。
- 处理各种请求的 Action。

14.16　在 Eclipse 中使用 JUnit 进行单元测试

在业务逻辑和 DAO 编写完成后需要对这些内容进行单元测试，这样就可以在后续的项目开发阶段减少错误的发生，提高工作效率。

JUnit 是一个比较优秀的测试工具，JUnit 在 Eclipse 中的配置较为简单，这里不再介绍。下面重点介绍在餐费管理系统中进行的单元测试。测试的类存在于 com.REP.test 路径下，主要针对自定义的 DAO 和业务逻辑进行测试。

14.16.1　测试 AccountDAO

AccountDAOTest：用于测试账户数据访问操作中是否存在问题，在这个测试中主要针对 AccountDAO 中的 GetAccountbyName()方法。代码如下：

```
public class AccoutDAOTest extends TestCase {
    AccountDAO accountdao = new AccountDAO();
    protected void setUp() throws Exception {
        //hibernate 初始化
        Configuration config = new Configuration().configure("com/REP/hbm/hibernate.cfg.xml");
        SessionFactory sessionfactory = config.buildSessionFactory();
        accountdao.setSessionFactory(sessionfactory);
    }

    /*
     * Test method for 'com.REP.DAO.AccountDAO.getAccountbyName(String)'
     */
    public void testGetAccountbyName() {
        Account     account =   accountdao.getAccountbyName("wlj");
        System.out.println("the balance is "+ account.getBalance());
    }

}
```

从代码可以看到，测试类 AccountDAOTest 继承自 TestCase 类，测试类中定义了两个方法，一个是 setUp()方法，一个是以 test 开头的 testGetAccountbyName()方法。

其中 setUp()方法是 TestCase 类中定义的方法，在子类中可以重写这个方法，这个方法用来完成测试所需的初始化工作。在每个自定义测试程序运行前，会先执行这个方法。

testGetAccountbyName()方法是自定义的测试方法，在定义时，如果方法名以 test 开头，运行测试程序时会自动运行这个方法。

运行 Junit 测试程序的方法和运行一般 Java 程序相同，在测试程序上右击，选择“Junit Test”,即可。测试如果发生错误将显示红色进度条提示，如图 14-21 所示，如果正确将显示出绿色进度条，如图 14-22 所示。

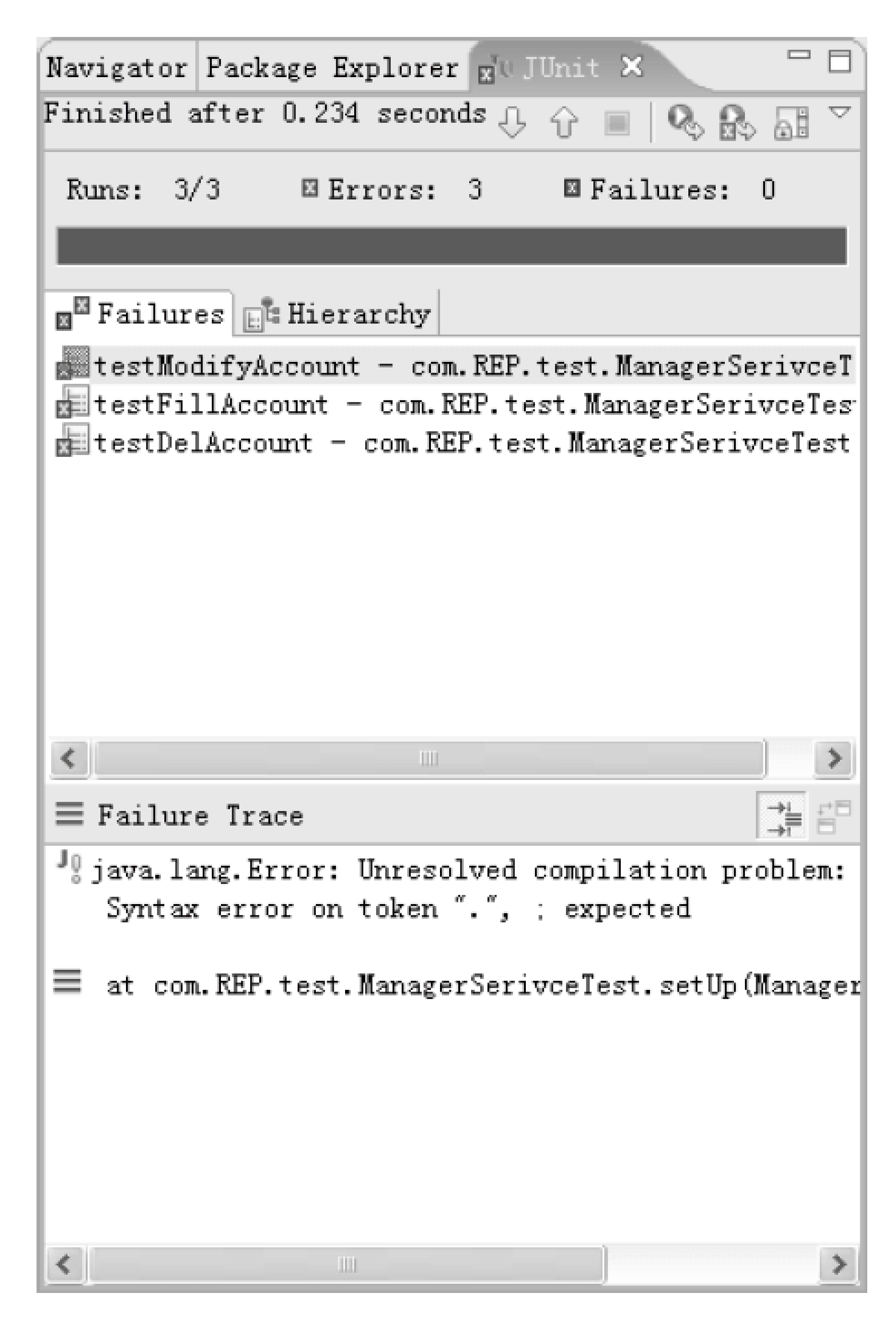



图 14-21　测试错误显示

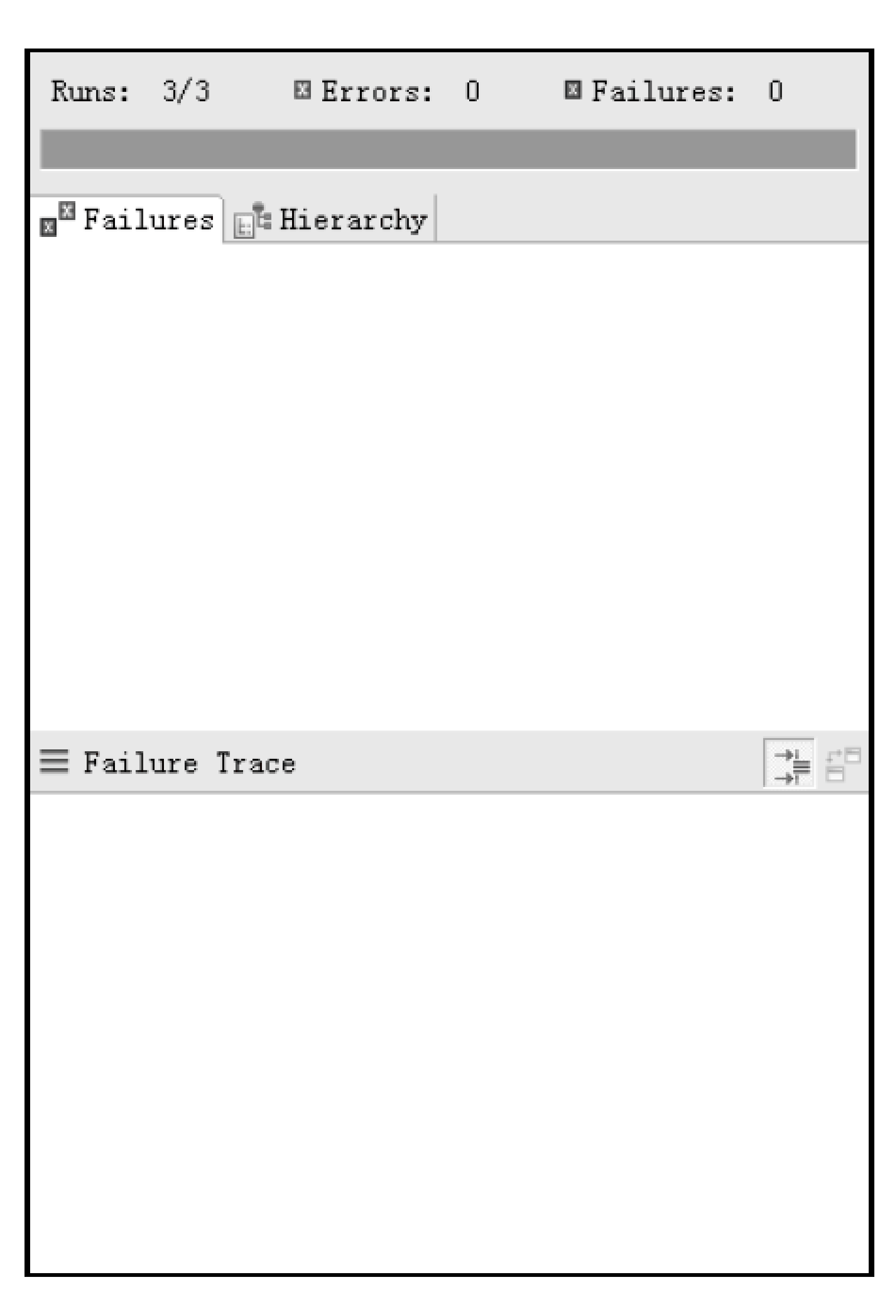


图 14-22　测试正确显示

14.16.2　测试 EmployeeDAO

EmployeeDAOTest：用来测试员工数据访问类 EmployeeDAO 中是否存在问题，在这个测试中主要实现了对保存员工和删除员工信息的测试。代码如下：

```
public class EmployeeDAOTest extends TestCase {
    public ManagerServiceImpl managerservice = new ManagerServiceImpl();
    public Account account = new Account();
    public AccountDAO accountdao =   new AccountDAO();
    public EmployeeDAO employeedao = new EmployeeDAO();
    protected void setUp() throws Exception {
        super.setUp();
        Configuration config = new Configuration().configure("com/REP/hbm/hibernate.cfg.xml");
        SessionFactory sessionfactory = config.buildSessionFactory();
        employeedao.setSessionFactory(sessionfactory);
        accountdao.setSessionFactory(sessionfactory);

    }

    /*
     * Test method for 'com.REP.DAO.base.BaseEmployeeDAO.save(Employee)'
     */
    public void testSaveEmployee() {
        Employee employee = new Employee();
        employee.setIdcard("1234567");
```

```
        employee.setName("ddd");
        employeedao.save(employee);
    }
    public void testFindEmployee(){
        Employee employee = employeedao.findEmployee("ddd","1234567");
        System.out.println(employee.getName());
    }
    public void testDelEmployee(){
        Employee employee = new Employee();
        employee.setIdcard("1234567");
        employee.setName("ddd");
        employeedao.save(employee);
        account.setLoginName("zs");
        account.setBalance(new BigDecimal("20"));
        account.setOverdrawNumber(2);
        account.setEmployee(employee);
        accountdao.save(account);
        employeedao.delete(employee);
    }

}
```

14.16.3　测试 EmployeeServiceImpl

EmployeeServiceTest：主要用来测试员工业务逻辑实现 EmployeeServiceImpl 是否存在问题，在测试之前首先初始化了一个就餐账户对象，然后针对这个账户对象进行注册（testRegist）和刷卡（testRepast）的测试。代码如下：

```
public class EmployeeServiceTest extends TestCase {

    public EmployeeServiceImpl employeeservice ;
    public Account account = new Account();
    public EmployeeDAO employeedao = new EmployeeDAO();
    public AccountDAO accountdao =   new AccountDAO();

    protected void setUp() throws Exception {
        super.setUp();
        //Hibernate 初始化
        Configuration config = new Configuration().configure("com/REP/hbm/hibernate.cfg.xml");
        SessionFactory sessionfactory = config.buildSessionFactory();

         //DAO 对象初始化
        employeedao.setSessionFactory(sessionfactory);
        accountdao.setSessionFactory(sessionfactory);

        //初始化员工就餐账户
        employeeservice = new EmployeeServiceImpl();
        employeeservice.setAccountdao(accountdao);
```

```
            employeeservice.setEmployeedao(employeedao);
            account.setLoginName("zs");
            account.setOverdrawNumber(2);
            account.setBalance(new BigDecimal("0"));

    }

    /*
     * Test method for 'com.REP.ServiceImpl.EmployeeServiceImpl.regist(Employee)'
     */
    public void testRegist() {
            try {
                    employeeservice.regist(account);
            } catch (AccountIsExistException e) {
                    System.out.println("account is already exist");
            }

    }

    /*
     * Test method for 'com.REP.ServiceImpl.EmployeeServiceImpl.repast(String, String)'
     */
    public void testRepast() {
        try {
            employeeservice.repast("zs","10");
        } catch (AccountNotExistException e) {
            System.out.println("the account is not exist");

        } catch (OverDrawException e) {
            System.out.println("you are overDraw");
        }
    }

}
```

14.16.4　测试 ManagerServiceImpl

ManagerServiceTest：用来测试管理员业务逻辑实现的正确性，在测试之前在 setUp()方法中初始化了一个员工（Employee）对象和一个账户（Account）对象，并分别把这两个对象持久化到数据库中，然后分别利用这两个对象进行了修改账户测试 testModifyAccount()、账户充值测试 testFillAccount()、删除账户测试 testDelAccount()。代码如下：

```
public class ManagerSerivceTest extends TestCase {
    public ManagerServiceImpl managerservice = new ManagerServiceImpl();
    public Account account = new Account();
    public AccountDAO accountdao =   new AccountDAO();
    public EmployeeDAO employeedao = new EmployeeDAO();
    protected void setUp() throws Exception {
```

```
        //Hibernate 初始化
        Configuration config = new Configuration().configure("com/REP/hbm/hibernate.cfg.xml");
        SessionFactory sessionfactory = config.buildSessionFactory();

        //在 DAO 对象中注入 SessionFactory 对象
        accountdao.setSessionFactory(sessionfactory);
        employeedao.setSessionFactory(sessionfactory);

        managerservice.setAccountdao(accountdao);

        //保存一个 Employee 对象
        Employee employee = new Employee();
        employee.setIdcard("111102197210080000");
        employee.setName("ddd");
        employeedao.save(employee);

        //保存一个账户对象
        account.setLoginName("zs");
        account.setBalance(new BigDecimal("20"));
        account.setOverdrawNumber(2);
        account.setEmployee(employee);
        accountdao.save(account);

    }

    /*
     * Test method for 'com.REP.ServiceImpl.ManagerServiceImpl.modifyAccount(Account)'
     */
    public void testModifyAccount() {

            try {
             account.setOverdrawNumber(3);
             managerservice.modifyAccount(account);
        }catch(Exception e){
             System.out.println("the account is not exist");
             e.printStackTrace();
        }

    }
    /*
     * Test method for 'com.REP.ServiceImpl.ManagerServiceImpl.fillAccount(Account, String)'
     */
    public void testFillAccount() {
            try {
             managerservice.fillAccount(account,"60");
        } catch (AccountNotExistException e) {
             System.out.println("the account is not exist");
        }
```

```
	}
	/*
	 * Test method for 'com.REP.ServiceImpl.ManagerServiceImpl.delAccount(Account)'
	 */
	public void testDelAccount() {
			try {
				managerservice.delAccount(account);
			} catch (AccountNotExistException e) {
				System.out.println("the account is not exist");
			}
	}
}
```

14.17　系统发布运行

启动 Web 服务器，这里用的是 Tomcat 服务器，在地址栏中输入如下内容：

```
http://localhost:8080/RepastExpensesManagement/
```

即可正常运行。